

目录

Neural Networks and Deep Learning.....	6
1 Deep learning intuition.....	6
1.1 Neuron neural network (自动学习特征)	6
1.2 神经网络种类.....	6
1.3 擅长非结构化数据.....	6
1.4 深度学习出现的根本原因.....	7
2 Logistic Regression as a Neural Network.....	7
2.1 intuition.....	7
1 图片分类 intuition.....	7
2 Derivative intuition.....	8
3 Computation graph and backpropagation.....	9
2.2 Logistics classification backpropagation.....	9
1 loss function 的由来(MLE).....	9
2 lr graph and gradient descent.....	11
3 vectorization+broadcasting.....	12
Vectorization.....	12
broadcasting.....	13
吴哥向量化建议.....	14
4 Coding 实现 logistics.....	15
小试牛刀改进一下.....	15
logistic 全面 vectorize! !	16
2.3 作业 notebook(vectorization+手工实现 logistics).....	18
3 Shallow Neural Network.....	18
3.1 Shallow Neural network 多个 logist 组合.....	18
3.2 activation function.....	19
regression 和激活函数.....	20
3.3 derivative of activation func.....	20
3.4 一个隐层神经网络 Vectorization 前向+后向推导(shape 约定)!.....	21
先约定参数 W 以及 X, A 的 shape.....	21
4 Deep Neural Network.....	24
4.1 Propagation.....	24
4.2 Hyperparameter control parameters.....	27
4.3 调节 hyper-parameter, 经验主义! !	27
4.4 deep learning feature detection.....	28
4.5 deep 比 large 更有效.....	28
Improving Deep Neural Networks Hyperparameter tuning, Regularization and Optimization.....	29
1 train/test/validate 划分注意.....	29
2 Bias / Variance 判定和解决(不同于传统 ML).....	29
3 Regularization(L1,L2,dropout)	31
3.1 regularization 本质理解.....	31
3.2 L1 Sparse.....	31

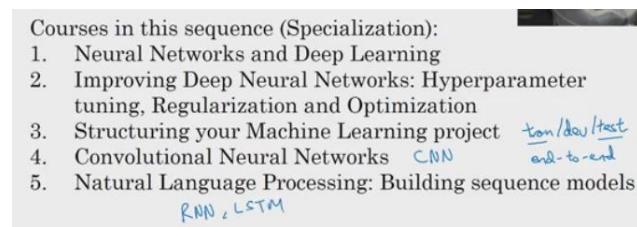
3.3 L2 weights decay.....	32
3.4 dropout(实现).....	33
4 其他 regularization 避免 overfitting 方案(代替 more data).....	35
4.1 Data Augmentation (regularzation tech).....	35
4.2 early stopping.....	35
5 训练原则: Orthogonalization.....	36
6. Prepare for optimization problem.....	36
6.1 Normalizing inputs 标准化(归一化)+本质.....	36
6.2 Vanishing / Exploding gradients.....	38
6.3 Weight Initialization for Deep Networks(not 0).....	38
6.4 Numerical approximation of gradients (gradient check)	39
7 Optimization algorithms	41
1. mini batch gradient descent.....	41
2 更多高级 Optimal algorithm.....	42
2.1 Exponentially weighted moving averages 指数加权平滑.....	42
2.2 Bias correction in exponentially weighted averages.....	43
2.3 Momentum.....	44
2.4 RMSprop Root mean square prop.....	45
2.5 Adam optimization algorithm(moment+rms)	46
2.6 Learning rate decay.....	47
2.7 The problem of local optima.....	48
8 Hyperparameter tuning.....	49
Tuning process.....	49
Coarse to fine 提炼参数.....	50
Using an appropriate scale to pick hyperparameters.....	50
Hyperparameters tuning in practice Pandas vs. Caviar.....	52
9 Batch Normalization.....	53
Normalizing activations in a network.....	53
Fitting Batch Norm into a neural network.....	54
Why does Batch Norm work.....	56
Batch Norm at test time.....	56
10 Multi-class classification.....	57
Softmax Regression.....	57
Training a softmax classifier.....	59
1 Cost func 定义.....	59
2 back propagation.....	60
11 Introduction to programming frameworks-tensorflow.....	60
Structuring Machine Learning Projects.....	63
1 大原则.....	63
2 Orthogonalization 垂直化.....	64
3 Single number evaluation metric 模型评估！	64
4 Satisficing + Optimizing metric(多 metric)	65
5 Train/dev/test.....	66
distributions.....	66

Size of the dev and test sets.....	67
6 When to change dev test sets and evaluate metrics.....	68
7 bias and variance based on human-level performance.....	69
Avoidable bias(结合 human level)	70
Understanding human-level performance.....	70
Surpassing human-level performance.....	71
Improving your model performance.....	72
8 error analysis.....	73
find error then improve.....	73
data augmentation.....	75
Cleaning up incorrectly labeled data(robust).....	75
Build your first system quickly, then iterate.....	77
Training and testing on different distributions.....	77
1 Bias and Variance with mismatched data distributions.....	78
2 Addressing data mismatch(可以 data augmentation)	80
9 Transfer learning.....	81
10 Multi-task learning.....	83
11 End-to-end deep learning.....	85
Convolutional Neural Networks(结构很重要，最好用经典结构！！才高准确率).....	86
1 computer-vision problem.....	86
2 filter 实现 feature detection.....	87
edge-detection-example.....	87
3 padding(对 input image padding, 避免边缘信息丢失).....	89
4 strided convolutions(完整计算公式).....	90
5 convolutions-over-volume 3D filter.....	91
5.1 3D filter.....	91
5.2 Learnable filter 本质.....	92
6 one-layer-of-a-convolutional-network(维度约定！).....	92
6.1 bias+non-linear activation.....	92
6.2 反向传播？	93
7 deep convolutional-network-example.....	95
8 pooling-layers.....	95
9 cnn-example LeNet-5(卷积网络的作用本质).....	96
10 why-convolutions.....	99
11 Resnets.....	99
11.1 Residual block: Skip the layer(一定见作业！).....	99
11.2 why-resnets-work.....	101
12 inception-network.....	102
12.1 1x1-convolutions.....	102
12.2 inception-network intuition.....	103
12.3 inception-network google net.....	105
13 practical-advice-for-using-convnets.....	106
13.1 using-open-source-implementation.....	106
13.2 transfer-learning.....	106

13.3 data-augmentation.....	107
13.4 state-of-computer-vision.....	108
14 object-detection.....	109
14.1 object-localization(detection).....	109
14.2 landmark(关键位置)-detection.....	111
14.3 object-detection.....	112
sliding window.....	113
用 convolutional-implement-sliding-windows.....	113
14.4 bounding-box-predictions YOLO(推出 slide window, stride).....	114
14.5 intersection-over-union IOU.....	116
14.6 Non max Suppression.....	117
14.7 anchor-boxes(one grid include several object center points).....	118
14.8 yolo-algorithm.....	120
14.9 region-proposals.....	122
15. face-recognition.....	123
15.1 intuition.....	123
15.2 one-shot-learning.....	123
15.3 Siamese-network 来实现相似函数(embedding).....	124
15.4 triplet-loss 定义 cost function.....	125
15.5 face-verification-转换为-binary-classification.....	127
16 neural-style-transfer 风格迁移	128
16.1 _what-are-deep-convnets-learning.....	128
16.2 cost-function.....	129
16.3 content-cost-function.....	130
16.4 style-cost-function.....	131
17 1d and 3d tensor generalizations.....	134
recurrent neural networks.....	136
1 recurrent neural networks.....	136
1.1 intition 和传统神经网络局限.....	136
1.2 recurrent-neural-network-model.....	137
Recurrent-neural-network representation.....	137
Forward propagation.....	139
backpropagation-through-time (question 如何 backpropagation)	140
different-types-of-rnns.....	140
1.3 language-model-and-sequence-generation.....	141
1.4 sampling-novel-sequences (hmm 和 RNN).....	143
1.5 Character level language model RNN.....	143
1.6 vanishing-gradients-with-rnns(gradients clipping)	144
1.7 gated-recurrent-unit GRU (RNN 的变形).....	144
1.8 long-short-term-memory-lstm (一定看作业).....	148
1.9 bidirectional-rnn BRNN.....	149
1.10 deep-rnns.....	150
1.11 coding 作业(rnn 的独特维度)和 loss 计算.....	151
2 natural-language-processing-word-embeddings.....	152

2.1 introduction-to-word-embeddings.....	152
word representation NLP and Word Embeddings.....	152
t-SNE algorithm 可视化.....	153
using-word-embeddings (trasfer learning).....	154
properties-of-word-embedding.....	154
embedding-matrix(其实就是 embeding 组成的 matrix).....	156
2.2 learning-word-embeddings.....	156
Neural language model.....	157
other target/context pair choose for learning embedding.....	157
word2vec (learn embedding 更简单有效).....	158
Skip-grams model (nearby word).....	158
Model detail.....	159
hierarhical 分层 softmax:	160
Negative-sampling 更有效的算法.....	160
解决随机选取频率高的 word 问题.....	161
glove-word-vectors(另一种方法).....	162
2.3 featurerization view of embedding.....	163
2.4 embedding 是需先单独 train from large text.....	164
2.5 applications-using-word-embeddings.....	164
1 sentiment classification.....	164
2 debiasing-word-embeddings.....	165
3 sequence-models-attention-mechanism.....	167
3.1 various-sequence-to-sequence-architectures.....	167
basic-models.....	167
machine translate.....	167
image caption.....	168
3.2 picking-the-most-likely-sentence.....	168
conditional language model.....	168
beam-search.....	169
refinements-to-beam-search.....	170
how to choose B.....	171
error-analysis-in-beam-search.....	171
bleu-score-optional.....	172
3.2 attention-model-intuition.....	174
3.3 speech-recognition-audio-data.....	179
speech-recognition(CTC).....	179
02_trigger-word-detection.....	180

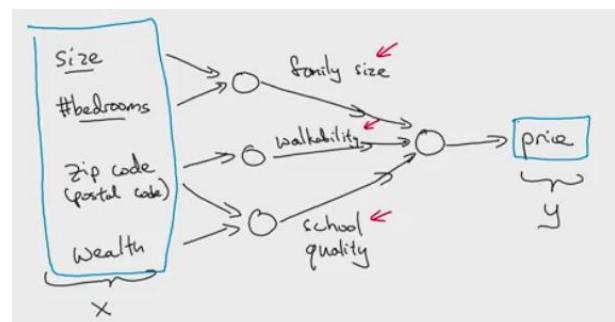
Neural Networks and Deep Learning



1 Deep learning intuition

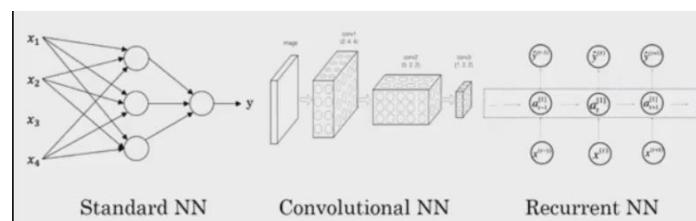
1.1 Neuron neural network (自动学习特征)

多个节点:



中间节点的具体含义，我们不需要管，神经网络会自动学习新的特征，自动生成学习 map 来 map x to y

1.2 神经网络种类



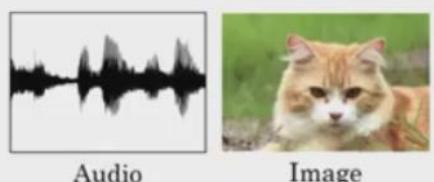
1.3 擅长非结构化数据

Structured data: 结构化数据

database date, 每个 feature 有固定的含义

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
:	:		:
27	71244		1

Unstructured data: 非结构化数据



Four scores and seven years ago...

Text

Deep learning 更擅长非结构化数据

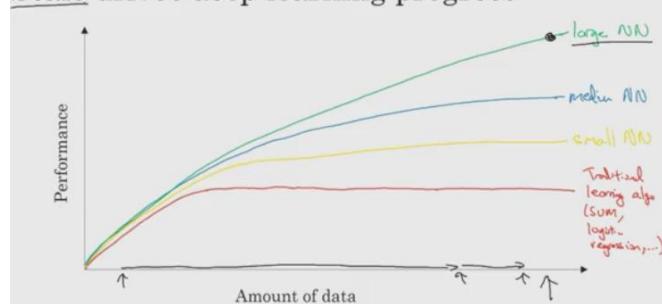
1.4 深度学习出现的根本原因

规模驱动了深度学习：

传统机器学习随着数据增长，表现平稳

而越复杂的神经网络，数据量越大，表现越好

Scale drives deep learning progress



激活函数：sigmoid, relu 计算 gradient 都很简单！！！

Run 更快，对于大网络。

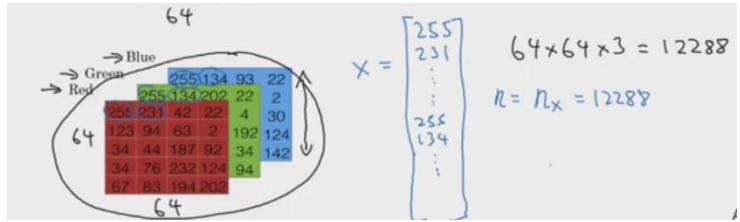
2 Logistic Regression as a Neural Network

2.1 intuition

1 图片分类 intuition

Unroll: 成 Input feature vector x

图片处理：



由 x 来预测 y

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(n)} \\ | & | & | \end{bmatrix}_{n \times m}$$

按列来排列 x 向量 矩阵(n*m)

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]$$

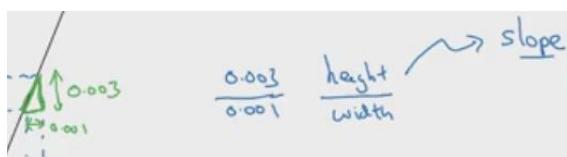
(1, m)

2 Derivative intuition

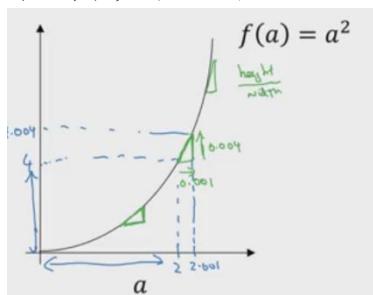
Derivative(导数)其实也是自变量的函数，代表不同自变量的函数变化率
所以需要带入 x 的值才知道该 x 的 gradient!

就是曲线或面的 slope 坡度(变化率)

Slope 定义：取一小区间的 y 和 x 变化量比值

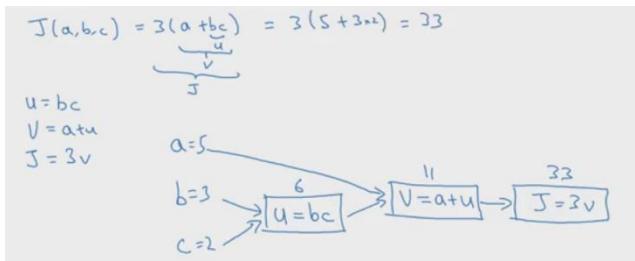


小三角形的 \tan 值

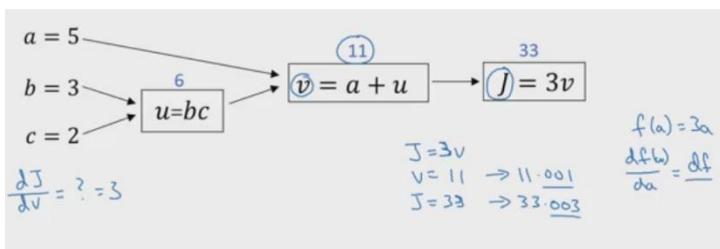


不同 slope 不同 derivative

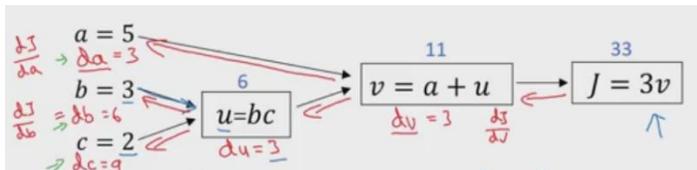
3 Computation graph and backpropagation



将复杂计算分解至简单步奏的计算，每个节点都是中间变量
 每一个节点的中间变量都可以计算关于 J 的导数，直到计算到 input 关于 J 的导数 $dJ/d(a, b, c)$



我们最终想求： dJ/da : a change--- v change--- J change
 $dJ/dv=3$
 $dv/da=1$
 chain rule: $dv/da= dJ/dv * dv/da$ 将上面的反过来



计算的关键在于：

从右到左遍历每个节点：1 先要计算最右边每个大节点（中间变量）的全局 gradient，如 dv du ，然后用来 2 计算该节点里的变量的全局 gradient dx ，3 需要用到该节点的局部 gradient

计算 2，需要 1 和 3， $1*3$ 得到 2

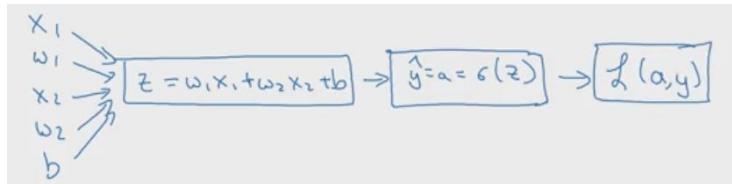
只需考虑三层就可以了，多层一次类推

中间变量全局 $gra * 目标对中间变量的局部 gra = 目标变量的全局$

2.2 Logistics classification backpropagation

1 loss function 的由来 (MLE)

就是单节点神经网络！！！



$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

如果用 Loss 如果用均方误差和, 就不是 convex 函数, 不能发现 global optimal, gradient descent 不能用均方误差和

cost func 的由来(如何定义一个 cost):

首先我们需要定义每个 example 的 loss

然后对于所有的 example, loss 求个并平均得到 cost function

首先对于一个 example:

首先 y hat, 我们的输出值是 $y=1$ 的概率 $p(y=1|x)$,

因此我们想当真实 label=1 时, 这个概率 $p(y=1|x)$ 最大。

当真实 label=0 时 $p(y=0|x)$ 最大: 其实也就是让 $p(y|x)$ 最大, 写成一个式子:

$$\begin{aligned} \text{If } y = 1: \quad p(y|x) &= \hat{y} \\ \text{If } y = 0: \quad p(y|x) &= 1 - \hat{y} \\ p(y|x) &= \hat{y}^y (1-\hat{y})^{(1-y)} \end{aligned}$$

MLE: choose the parameter maximize the term (不一定是 $\max p(x)$, 根据实际问题建模) (parameter 在 y hat 里)

$\max p(y|x)$ 等价于 $\max \log p(y|x)$ 等价于 $\min -\log p(y|x)$

因此 $-\log p(y|x)$ 就是我们的 loss! ! !

$$\begin{aligned} \log p(y|x) &= \log \hat{y}^y (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y) \log (1-\hat{y}) \\ &= -L(\hat{y}, y) \end{aligned}$$

Andrew Ng

以上是一个 example 的 loss func

其实就是:

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$L(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

多个 example: 假设 IID, 每个 example 独立出现, 也就是使得乘积最大

加上 log 就是和! 然后在 min 负值

choose 参数, 来 $\max \log p$

$$\begin{aligned}
 \log p(\text{labels in training set}) &= \log \prod_{i=1}^m p(y^{(i)} | x^{(i)}) \\
 \log p(\dots) &= \sum_{i=1}^m \underbrace{\log p(y^{(i)} | x^{(i)})}_{-\mathcal{L}(y^{(i)}, \hat{y}^{(i)})} \\
 &= -\sum_{i=1}^m \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) \\
 (\text{Cost: minimize}) \quad J(w, b) &= \frac{1}{m} \sum_{i=1}^m \mathcal{L}(y^{(i)}, \hat{y}^{(i)})
 \end{aligned}$$

cost func 其实就是所有 example 的 loss sum 的均值

因此 dJ/dw , 是每个 example 的 $d \text{ loss}/dw$ 的和的均值, 因为每个 example 的 loss func 都有 w 和 x 的内积

2 lr graph and gradient descent

后向传播; 是为了计算每个参数的全局 gradient! 用于更新参数, 使得参数以 minimize loss 的方向变化 (梯度下降)

不需要记节点, 只需要按照嵌套的函数, 从外到内把所有中间变量的全局 gradient 计算出, 计算当前变量需要 1 当前变量上层函数的局部 gradient 2 上层函数的全局 gradient , 没有节点这一说! d 和不加 d 的维度一致

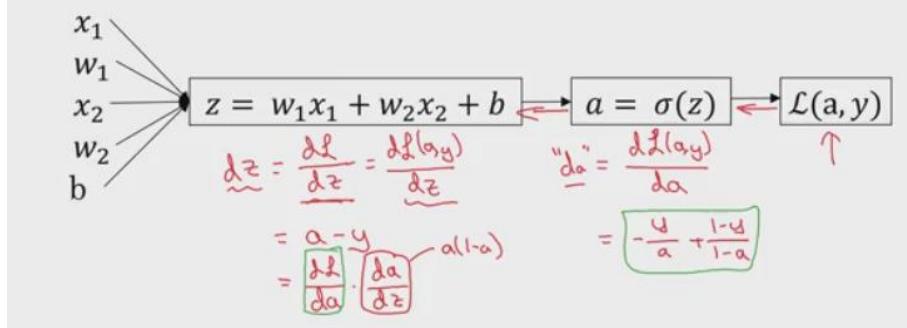
对于一个 example:

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

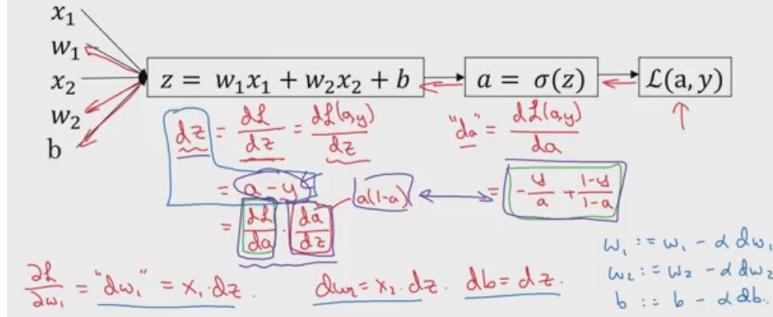
Logistic regression derivatives



从右到左:

先求 da 全局 gradient ; 求 dz 全局=da*dz 局部等于 a-y
 dw db 等于 $dz*w/b$ 的局部 : w 局部是 x, b 局部是 1
 dw 是向量了也就是 x 向量！！

Logistic regression derivatives



对多个 example 进行 gradient descent, J 的 gradient, 其实只是在上面的基础上将每个 loss 的 gradient 的累加并求均值

1 J cost func 是针对所有的 example(所有 example 的 loss 累加均值)

2 每个参数的 gradient 是对 J 的, 因此, dJ/dw 是 所有 example 的 dL/dw 的累加均值, 是用这个来更新参数！！

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m l(a^{(i)}, y^{(i)})$$

因为 Cost 的定义就是所有 example 的 loss 的 sum, 因此求 w_1 的 gradient, 每个 example 都有 w_1 的 gradient, 因此 Cost 的 gradient 需要 sum

dJ/dw_1 就是 sum 并平均每个 example 的 loss gradient

3 vectorization+broadcasting

Vectorization

包括矩阵运算+向量元素级操作！！能用向量的尽量用向量

np 里的向量函数 np.exp(x) 等等, x 是向量, 效率非常高！！！
 代替 loop！！！

```
x = np.array([1, 2, 3])
print(np.exp(x)) # result is (exp(1), exp(2), exp(3))
[ 2.71828183  7.3890561  20.08553692]
```

1 矩阵运算

直接矩阵运算：效率更高

np. dot (w, x)

w, x 是两个 np 数组

```
Vectorized version: 1.5027523040771484ms  
250286.989866  
For loop: 474.29513931274414ms
```

效率相差太大了

可以利用 GPU CPU 多线程作用

2 元素级的操作 element wise 操作，也比循环高

尽量使用 numpy 的元素级函数

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

`import numpy as np
u = np.exp(v)`

$\Rightarrow u = np.zeros((n, 1))$
 $\Rightarrow for i in range(n):$
 $\quad \Rightarrow u[i] = math.exp(v[i])$

右边比左边效率高， np 内置函数

包括 两个向量的 element wise 相乘得到向量也算是 vectorize

更不用说矩阵运算

都比遍历更高效！！！

尽量用以 vector 为单位的函数操作！！！

broadcasting

定义矩阵 A:

```
[[ 56.      0.      4.4    68. ]  
 [ 1.2     104.     52.     8. ]  
 [ 1.8     135.     99.     0.9]]
```

```
cal = A.sum(axis=0)  
print(cal)  
[ 59.   239.   155.4  76.9]
```

```
percentage = 100*A/cal.reshape(1,4)  
print(percentage)  
[[ 94.91525424  0.          2.83140283  88.42652796]  
 [ 2.03389831  43.51464435  33.46203346  10.40312094]  
 [ 3.05084746  56.48535565  63.70656371  1.17035111]]
```

Sum axis=0 按列加，得到一维数组

A (3,4) cal(1,4) cal 会扩展到(3,4) 再元素级相除

向量和实数：实数扩展

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} = \begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$$

矩阵和向量：向量扩展

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{(m,n)} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix}_{(l,n) \rightsquigarrow (m,n)} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}_{(l,n)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{(m,n)} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix}_{(m,1) \downarrow (m,n)} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}_{(l,n)}$$

缺哪一维度就在哪一维扩展

吴哥向量化建议

向量化时，不要用一维数组 (5,) np.random.randn(5)
一维数组转置.T 还是本身

```
import numpy as np
a = np.random.randn(5)

print(a)
[ 0.50290632 -0.29691149  0.95429604 -0.82126861 -1.46269164]

print(a.shape)
(5,)

print(a.T)
[ 0.50290632 -0.29691149  0.95429604 -0.82126861 -1.46269164]

print(np.dot(a,a.T))
4.06570109321
```

本来结果我们期望是 n*n 结果成了内积

向量化一定要用列向量(属于二维矩阵范畴)

np.random.randn(5,1) 列向量

```

a = np.random.randn(5,1)
print(a)

[[-0.0967311 ]
 [-2.38617377]
 [-0.3243588 ]
 [-0.96216349]
 [ 0.54410384]]
```

```

print(a.T)

[[-0.0967311 -2.38617377 -0.3243588 -0.96216349  0.54410384]]
```

```

print(np.dot(a,a.T))

[[ 0.00935691  0.23881721  0.03137558  0.09307113 -0.05263176]
 [ 0.23081721  5.69382526  0.77397645  2.29588928 -1.2983263 ]
 [ 0.03137558  0.77397645  0.10520863  0.31208619 -0.17648487]
 [ 0.09307113  2.29588928  0.31208619  0.92575858 -0.52351684]
 [-0.05263176 -1.2983263  -0.17648487 -0.52351684  0.29684898]]
```

一旦出现一维数组，需要 reshape(5,1)

np. dot 操作

所以最好都转成二维的！！！统一，减少错误

```

A=np.zeros((1,2))
x=np.zeros((2,4))
print(np.dot(A, x))

[[ 0.  0.  0.  0.]]
```

Suppose img is a (32,32,3) array, representing a 32x32 image with 3 color channels red, green and blue. How do you reshape this into a column vector?

- x = img.reshape((1,32*32,*3))
- x = img.reshape((3,32*32))
- x = img.reshape((32*32,3))
- x = img.reshape((32*32*3,1))

4 Coding 实现 logistics

小试牛刀改进一下

gradient 变量名的表示全局 gradient 就是 dx

迭代法：

Logistic regression on m examples

$$\begin{aligned}
 J &= 0; \underline{dw_1} = 0; \underline{dw_2} = 0, \underline{db} = 0 \\
 \text{For } i &= 1 \text{ to } m \\
 z^{(i)} &= w^T x^{(i)} + b \\
 a^{(i)} &= \sigma(z^{(i)}) \\
 J_t &= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})] \\
 \underline{dw_1}^{(i)} &= a^{(i)}(1-a^{(i)}) \\
 \underline{dw_2}^{(i)} &= x^{(i)} \underline{dw_1}^{(i)} \\
 db^{(i)} &= a^{(i)} \\
 J/m &\leftarrow \\
 dw_1/m &= \underline{dw_1}^{(1)} + \dots + \underline{dw_1}^{(m)} / m \\
 dw_2/m &= \underline{dw_2}^{(1)} + \dots + \underline{dw_2}^{(m)} / m \\
 db/m &= \underline{db}^{(1)} + \dots + \underline{db}^{(m)} / m
 \end{aligned}$$

迭代法求，遍历 m 求得 gradient 再更新参数那个参数
先前向传播，后后向传播(m 次累加 dw_1 , dw_2)最后算均值

Vectorization 向量化！可以应用在大数据上！！！

dw 变量向量化

$dw = np.zeros(n, 1)$

Logistic regression derivatives

$$\begin{aligned}
 J &= 0, \quad \boxed{dw_1 = 0, dw_2 = 0}, \quad db = 0 \quad dw = np.zeros((n_x, 1)) \\
 \rightarrow \text{for } i &= 1 \text{ to } n: \\
 z^{(i)} &= w^T x^{(i)} + b \\
 a^{(i)} &= \sigma(z^{(i)}) \\
 J &+= -[y^{(i)} \log \hat{a}^{(i)} + (1-y^{(i)}) \log(1-\hat{a}^{(i)})] \\
 \frac{\partial J}{\partial w_1} &= x^{(i)} \frac{\partial \hat{a}^{(i)}}{\partial z^{(i)}} \quad n_x = 2 \quad dw_1 += x^{(i)} dz^{(i)} \\
 \frac{\partial J}{\partial w_2} &= x^{(i)} \frac{\partial \hat{a}^{(i)}}{\partial z^{(i)}} \quad dw_2 += x^{(i)} dz^{(i)} \\
 db &+= \frac{\partial J}{\partial b} \\
 J &= J/m, \quad \boxed{dw_1 = dw_1/m, dw_2 = dw_2/m}, \quad db = db/m \\
 dw &/= m
 \end{aligned}$$

上面 m 写成 n 了！！

上面计算 dw 的方法其实也是遍历，我们改成 vectorize，两个向量 x 是向量 dz 是向量 elementwise 相乘，也能提高效率

上面只是很小的一个改进，下面才是玩真的！！！！下面高能！！！

logistic 全面 vectorize！！

因为有了 m ，至少都是向量，有些变成矩阵了

先定义好维度：大写矩阵 小写向量

每个 example 定义成向量，每个 z 和 a 是行向量， w 是向量 (logistics 的 weight 是向量！！！)

1 向前传播： X 到 a

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix} \quad (n_x, m)$$

$$\underline{z}^{(1)} = w^T x^{(1)} + b$$

$$\underline{a}^{(1)} = \sigma(z^{(1)})$$

$$[z^{(1)} z^{(2)} \dots z^{(m)}] = w^T X + [b \ b \dots b]_{1 \times m}$$

$X(n, m)$ 矩阵 $w(n, 1)$ 列向量

$z = w \cdot T * X \quad (1, m)$ 每个 example 一个激活值 z 和 a 是行向量

$b(1, m)$ 也是行向量

得到 z 的向量化的公式，然后实现：

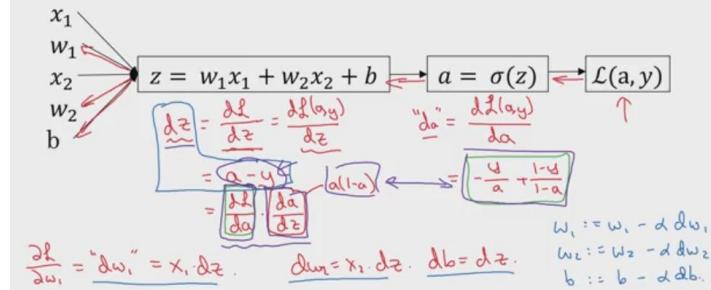
$$z = np.dot(w.T, X) + b \in (1, 1) \quad \mathbb{R}$$

broadcasting b 是实数，和向量相加会先转成向量

然后 $\text{sigmoid}(z)$ 得到 a 向量函数运算

2 后向传播：一个 example 为例

Logistic regression derivatives



根据前面约定，先推导出公式！！！！直接根据公式计算

首先 cost(cross entropy) 是 a 中间变量的函数，求出 da
 $dz = da * dz = a - y$

$dz = a - y$ 向量化 $(1, m)$

$$dz^{(1)} = a^{(1)} - y^{(1)} \quad dz^{(2)} = a^{(2)} - y^{(2)} \quad \dots$$

$$dz = [dz^{(1)} \ dz^{(2)} \ \dots \ dz^{(m)}]_{1 \times m}$$

$$A = [a^{(1)} \ \dots \ a^{(m)}], \quad Y = [y^{(1)} \ \dots \ y^{(m)}]$$

$$dz = A - Y = [a^{(1)} - y^{(1)} \ a^{(2)} - y^{(2)} \ \dots]$$

有了前面的 dz ，就不需要管后面的 da 了，继续往前推 $dw \ db$ 也是向量：

$dw = x dz$ dw 是向量

向量化公式：

$X(n, m) \ dz(1, m) \ dw(n, 1)$ 向量

$dw = X * dz \cdot T$

(每个 m 长的 vector 两两内积(先 element 相乘，再 sum)，最后得到 n 个元素)

$$\delta w = \frac{1}{m} \times \delta z^T$$

$\delta b = \delta z$

每个 example 的 $\delta b_i = \delta z_i \quad \delta z (1, m)$

更新 δb 只需将 m 个 example 相加求均值就好

$$\begin{aligned}\delta b &= \frac{1}{m} \sum_{i=1}^m \delta z^{(i)} \\ &= \frac{1}{m} \underbrace{\text{np.sum}(\delta z)}\end{aligned}$$

最终 gradient 公式：

$$\begin{aligned}\frac{\partial J}{\partial w} &= \frac{1}{m} X(A - Y)^T \\ \frac{\partial J}{\partial b} &= \frac{1}{m} \sum_{i=1}^m (a^{(i)} - y^{(i)})\end{aligned}$$

$$w := w - \alpha \delta w$$

$$b := b - \alpha \delta b$$

推出公式就好办了！！！矩阵，向量化运算

这是一次训练，更新参数，需要多个 epoch

定义好维度！！！

我们可以看出，求 gradient 都需要 Z, A, Y 已知，也就是每前向传播一次，就要记录这些值，然后后向传播的时候会用到！！

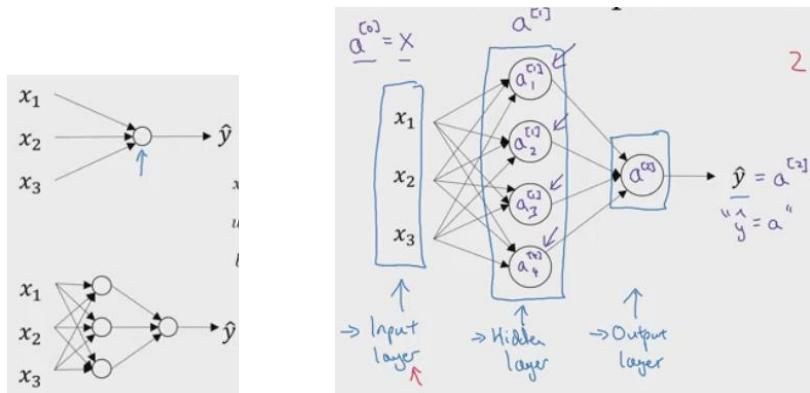
一直 repeat：前向传播 → 后向传播 前向传播 → 后向传播

前向传播和后向传播是一次传播（一次更新参数），后向传播需要用到前向传播的计算的中间变量！！！

2.3 作业 notebook (vectorization+手工实现 logistics)

3 Shallow Neural Network

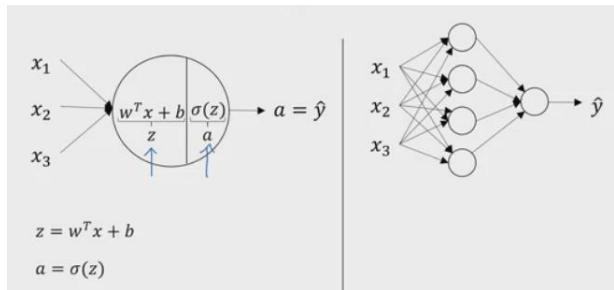
3.1 Shallow Neural network 多个 logist 组合



layer 不算 input layer

上标代表层，下标代表第几个节点

3.2 activation function

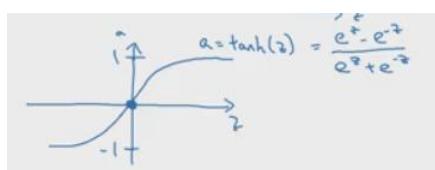


每个 note 都是两步计算; 1 计算 z 2 计算激活值

sigmoid 是 0-1 tanh 是 -1-1

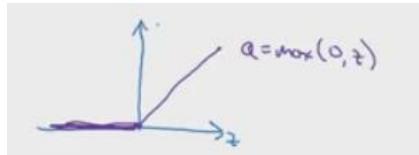
tanh function 比 sigmoid work 的更好

使每一层输出都是 zero mean, perform better!



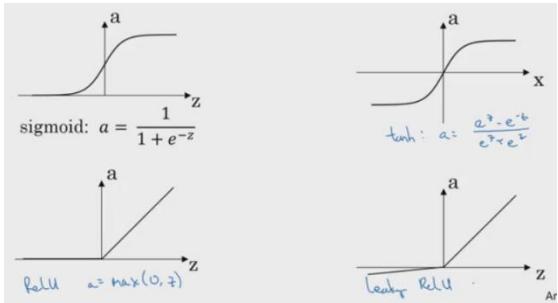
但对于二分类，output 一般用 sigmoid func，
tanh 和 sigmoid 缺点：当 z 很大时，gradient 接近 0，会使得 back propagation 变慢

relu func 最好的 default activation func



但当 z 为负时，gradient 为 0，但大多数情况下 z 都是大于 0，能够很快的后向传播

leaky Relu



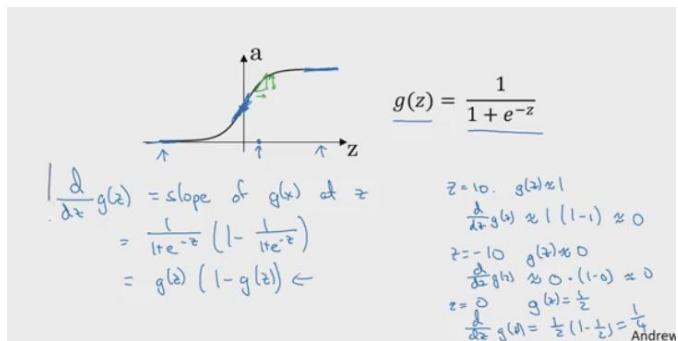
为什么要用激活函数？？

不用的话，只是 linear classifier of input

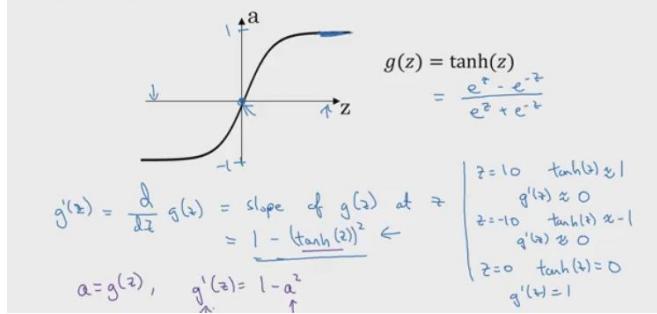
regression 和激活函数

当 output 是做 regression 时，一般不用 activation function 或有时候我们的 output 是房价，不能是负的，可以用 relu

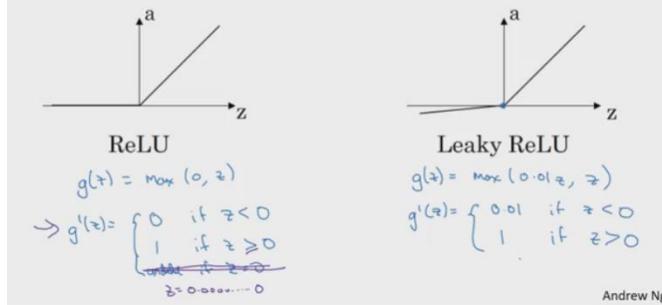
3.3 derivative of activation func



Tanh activation function



ReLU and Leaky ReLU



The ReLU activation function can help prevent vanishing gradients.

Dead ReLU Units

Once the weighted sum for a ReLU unit falls below 0, the ReLU unit can get stuck. It outputs 0 activation, 使得该 node 的 weight 的 gradient 成为 0, gradients can no longer flow through it during backpropagation. With a source of gradients cut off, the input to the ReLU may not ever change enough to bring the weighted sum back above 0.

Lowering the learning rate can help keep ReLU units from dying.

3.4 一个隐层神经网络 Vectorization 前向+后向推导 (shape 约定) !

推导都是以 1 个 example 为推导, 然后改成多 example

因为计算 loss 都是用一个 example, 计算 loss 的导数也是一个 example(函数的形式也是一个 example, 因此求导也这样), 改成多个 example 只不过是为了 gradient 求和, 更新参数!!!

因此我们只需关注于一个 example 的推导!!!得到推导公式, 再改成多个 example

先约定参数 W 以及 X, A 的 shape

每一层一个参数矩阵 W :

上标代表第 i 层, 下标第 i 个 node

1 对于 1 个 example 的 shape:

$x(n, 1)$ $W(k, n)$ $W*x = a (k, 1)$ $b(k, 1)$

x, a 都是列向量, W 是矩阵(当前层节点个数, 前一层 input 个数)

先用 1 个 example 的去推导公式, 写出公式

2 对于多个 example 的 shape:

A, Z, X, b 第二个维度才是 m 只是多加了个 m 维度而已

W 只是前一层后一层的矩阵

1 $X(n, m)$ W(当前层节点个数, 前一层 input 个数)

其实 X 和 W shape 是一致的, 可以理解为 (feature 个数, m)

对于 W 来说, 前一层的 input 个数就是当前层的 m! ! !

X 每个 example 一个列向量, W 该层的参数是一个列向量

2 WX (当前层节点个数, m) 也是 A 和 Z 的 shape 和 X, W 也是一致的
也就是 A 每一层激活值作为向量

2.1a 和 z 如果是多个节点, 和 X 排列一致, 如果只有一个节点就是行向量!!!
向前传播:

3 b 是每个节点一个 b (b 对于 m 都一样), b 只需定义成 (当前层节点个数, 1)
列向量!

对于前面的公式, 直接改成我们多个 example 的维度!!

1 前向传播推导:

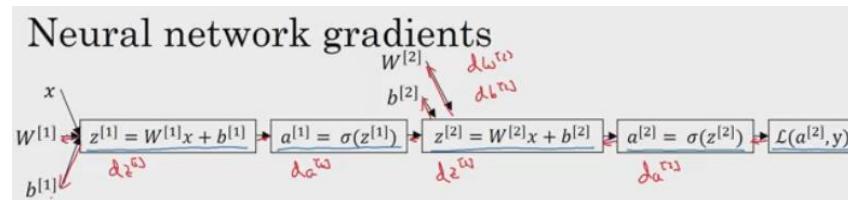
$z = Wx + b$ 改成多 example, 直接改, 顺序都不变!!

$$\begin{aligned} z^{[1]} &= W^{[1]T} X + b^{[1]} \\ A^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= W^{[2]T} A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(z^{[2]}) \end{aligned}$$

2 后向传播:

推导公式:

一点一点向后传播, 计算 gradient



要从后往前计算出所有中间变量的 gradient!!

$$\begin{aligned} z^{[2]} &= W^{[2]}x + b^{[2]} \\ \text{d}z^{[2]} &= \text{d}a^{[2]} - y \\ \text{d}W^{[2]} &= \text{d}z^{[2]} a^{[1]T} \\ \text{d}b^{[2]} &= \text{d}z^{[2]} \end{aligned}$$

首先知道 $dz[2] = a[2] - y$

1 推出第二层的参数 $dw[2] = dz[2] \cdot (中间变量全局) * x$ (该参数的局部)

此时 x 是 $a[1]$

两个乘子都列向量，需要一个转置，求外积，得到矩阵

$$db[2] = dz[2]$$

2 求出一层的参数 gradient 后，求下一层的中间变量全局 gradient $da[1]$

求 $a[1]$ 的 gradient

$da[1] = dz[2] \cdot (中间变量全局) * W[2]$ ($a[1]$ 的局部)

$dz[2]$ (2 层节点数, m) $W[2]$ (2 层节点数, 1 层节点数)

相乘是 $da[1] = W[2].T * dz[2]$ (1 层节点数, m)

求 dz 的 gradient

$dz[1] = da[1] \cdot (中间变量全局) * (z[1])$ (该变量的局部)

总结：

$$\begin{aligned} dz^{[2]} &= a^{[2]} - y \\ dW^{[2]} &= dz^{[2]} a^{[1]T} \\ db^{[2]} &= dz^{[2]} \\ dz^{[1]} &= W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]}) \\ dW^{[1]} &= dz^{[1]} x^T \\ db^{[1]} &= dz^{[1]} \end{aligned}$$

规律：求某个变量的全局 gradient=它上层中间变量的全局 * 该变量的局部
这个变量也可能是中间变量(多层神经网络)

改成多个 example:

因为之前求 dW 只是向量外积，现在是两个矩阵相乘，已经完成了 m 求和的过程，
所以只需多加个 $1/m$ ，同理 db 也要求均值

其他公式不变

Summary of gradient descent

$$\begin{array}{ll} dz^{[2]} = a^{[2]} - y & dZ^{[2]} = A^{[2]} - Y \\ dW^{[2]} = dz^{[2]} a^{[1]T} & dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T} \\ db^{[2]} = dz^{[2]} & db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True) \\ dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]}) & dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]}) \\ dW^{[1]} = dz^{[1]} x^T & dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T \\ db^{[1]} = dz^{[1]} & db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True) \end{array}$$

Andrew Ng

左边是一个 example 情况，右边是多个 example 向量化

先写出左边，再向量化

$$\begin{aligned}
 dZ^{[2]} &= A^{[2]} - Y \\
 dW^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]T} \\
 db^{[2]} &= \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True) \\
 dZ^{[1]} &= \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[2]}, m)} * \underbrace{g^{[1]'}(Z^{[1]})}_{\text{elementwise product}} \\
 dW^{[1]} &= \frac{1}{m} dZ^{[1]} X^T \\
 db^{[1]} &= \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)
 \end{aligned}$$

Andrew Ng

我们可以看出，求 gradient 都需要 Z, A, Y 已知，也就是每前向传播一次，就要记录这些值，然后后向传播的时候会用到！！

一直 repeat：前向传播—> 后向传播 前向传播—> 后向传播

前向传播和后向传播是一次传播(一次更新参数)，后向传播需要用到前向传播的计算的中间变量！！！

如果是 batch gradient descent, iteration 就是传播的次数！！！ for loop

4 Deep Neural Network

4.1 Propagation

多个隐层！！！ L 是层数 n[i] 是每一层 unit 数量

a[i] w[i] 每一层 activation weight

需要一个 dict 来组织，key 是层数，value 是变量值

定义维度；

W(当前层 unit 数量, 前一层 unit 数量)

b(当前层 unit 数量, 1) 列向量

$$\begin{aligned} \rightarrow \omega^{[2]} &: (n^{[2]}, n^{[1-1]}) \\ \rightarrow b^{[2]} &: (n^{[2]}, !) \\ \delta\omega^{[2]} &: (n^{[2]}, n^{[1-1]}) \\ \delta b^{[2]} &: (n^{[2]}, !) \end{aligned}$$

对于多个 example:

A, Z, X, b 第二个维度才是 m 只是多加了个 m 维度而已

$$Z^{(i)} = W^{(i)} \cdot X + b^{(i)}$$

$\overbrace{(n^{(i)}, m)}^{\uparrow} \quad \overbrace{(n^{(i)}, n^{(i)})}^{\uparrow} \quad \overbrace{(n^{(i)}, m)}^{\leftarrow} \quad \overbrace{(n^{(i+1)}, 1)}^{\leftarrow}$

1 向前传播：

左边是 1 个 example，右边是 vectorize，核心就是维度要对的上！！！

Forward propagation for layer l

$$\begin{aligned} &\rightarrow \text{Input } a^{[l-1]} \leftarrow \underbrace{W^{[l-1]} b^{[l-1]}}_{\text{Cache}} \\ &\rightarrow \text{Output } a^{[l]}, \text{cache } (\underline{z}^{[l]}) \\ z^{[l]} &= W^{[l]} \cdot a^{[l-1]} + b^{[l]} \\ a^{[l]} &= g^{[l]}(z^{[l]}) \end{aligned} \quad \left| \begin{array}{l} \text{Value:} \\ z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]} \\ A^{[l]} = g^{[l]}(z^{[l]}) \end{array} \right.$$

前向传播函数：只需要输入前面一层的 activation，输出后面一层的 activation，
cache 中间变量 z

$$Z^{[l]} = W^{[l]} * A^{[l-1]} + b^{[l]}$$

$$Z(k_l, m) = W(k_l, k_{l-1})^* A(k_{l-1}, k_{l-2}) + b(k_l, 1)$$

需要遍历每一层！！！ X 统一成 A[0]

2 向后传播

Backward propagation for layer l

→ Input $da^{[l]}$
 → Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$dz^{[l]} = da^{[l]} \times g'(z^{[l]})$ $dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$ $db^{[l]} = dz^{[l]}$ $da^{[l-1]} = W^{[l]} \cdot dz^{[l]}$ $dz^{[l]} = W^{[l-1] T} \cdot dz^{[l-1]} + g'(z^{[l]})$	$dz^{[l]} = \Delta A^{[l]} \times g'(z^{[l]})$ $dW^{[l]} = \frac{1}{m} \sum_{i=1}^m dz^{[l]} \cdot a^{[l-1] T}$ $db^{[l]} = \frac{1}{m} \sum_{i=1}^m dz^{[l]}$ $dA^{[l-1]} = W^{[l-1] T} \cdot dz^{[l]}$
--	--

对于每一层的最外层函数是 activation，因此起始 gradient 是 $da[1]$

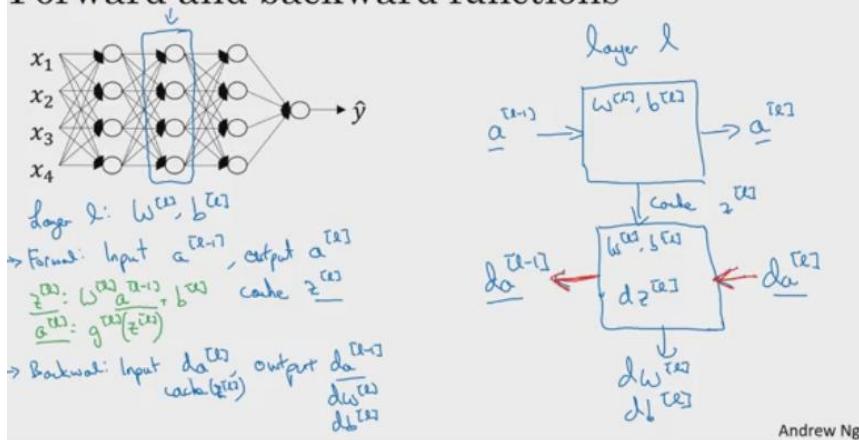
后向传播函数：输入 L 层的 da ，然后计算: L 的 dW 和 db , L-1 层 da

如上图

- 有了 L 层 da ，可以求 L 层 dz ，先求局部 $da/dz=g(z)(1-g(z))$ ，因此需要 L 层 z 值 (forward 时 cache)
- 有了 dz 就可以求 dW , db 和 L-1 层的 da
 L-1 层的 da : L-1 层的 a 就想到于 x , 相当于求 dx , 其实就是 W

一层一层的这样计算

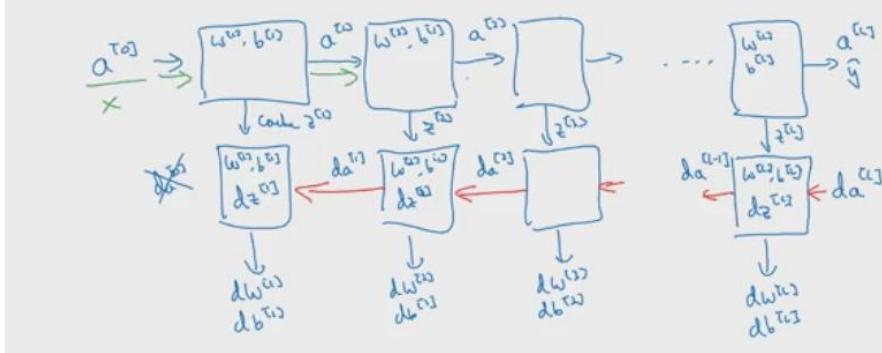
Forward and backward functions



左边定义了 forward 和 backward function 输入输出和传输关系
 forward cache z

backward 需要用到 W b ，会计算出 dz dW db $da[1-1]$ ，输出 dW db , $da[1-1]$

Forward and backward functions



执行一遍 forward, cache 每一层的 z

然后执行 backward, 计算每一层的参数 gradient, 和 da 给下一层用
每个 block, 都是一个方法

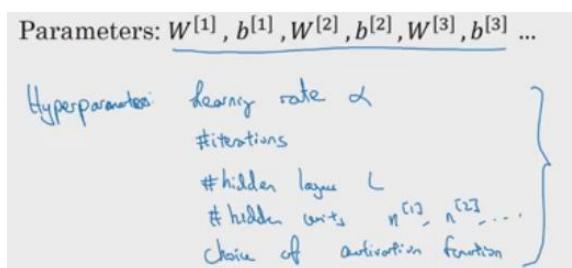
每一个 dw db 累加 m 次然后除以 m, 再更新参数！！！

先推导写公式, 确定好每个量的维度, 只要公式推导写出来了 一切就按照公式来！！！！

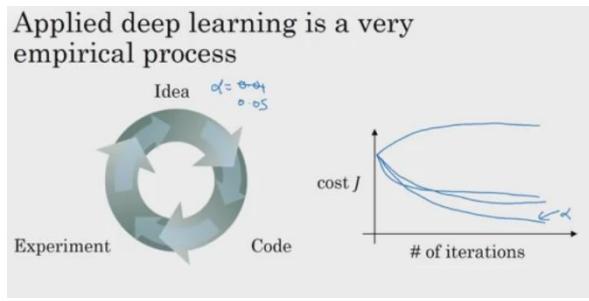
```
#每一层的 linear cache W b A 用来计算 linear 的 gradient
#计算 dA db dW 需要用到 linear cache, 因为都是线性的部分
#每一层的 activation cache Z 计算 activation gradient
#计算 dZ 需要用到 linear cache Z, 因为都是 activation 的部分
```

4.2 Hyperparameter control parameters

上面是 parameter, 下面是 hyper-parameter
hyper-parameter 会控制 parameter 的值

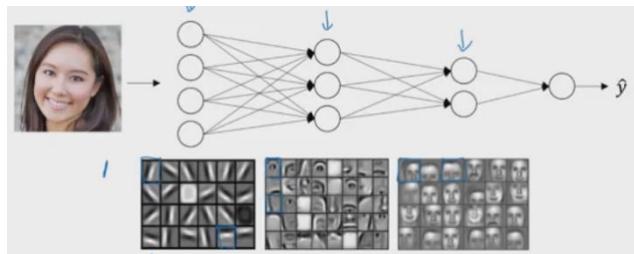


4.3 调节 hyper-parameter, 经验主义！！



4.4 deep learning feature detection

第一层很小的局部，后面是复杂的组合



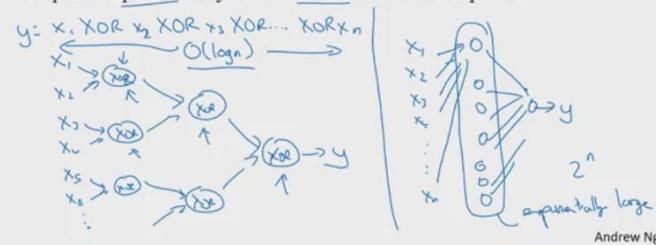
Audio---也是一样，从简单的 feature，随着越深，学到的 feature 越复杂！！

4.5 deep 比 large 更有效

deep net 等价于 大指数倍的 network
 deep 更有效！

Circuit theory and deep learning

Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.



Improving Deep Neural Networks Hyperparameter tuning, Regularization and Optimization

1 train/test/validate 划分注意

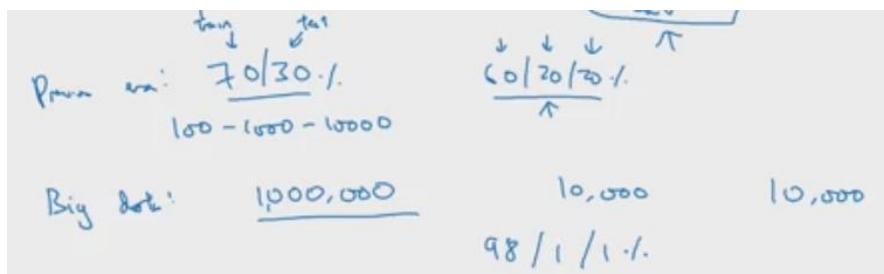
ML 是个 iterative process 迭代过程

需要不断尝试：如何高效迭代：



不同的数据量，分割比例不同

小的数据，比例可以传统 6-2-2



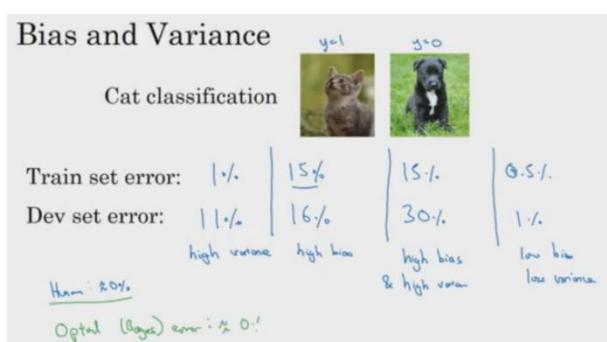
大数据，test 和 validate(development) 不用太多

mismatch train test distribution

高分辨率 vs 低分辨率

一定要保证 dev 和 test from same distribution! !!

2 Bias / Variance 判定和解决(不同于传统 ML)



3 个指标判断 bias 和 variance

1 Train 上表现好，dev 上表现不好，过拟合

2 表现都不好，bias，欠拟合，模型没有捕捉到 data pattern

2 表现都不好，但 train 上好过 dev, 既 bias 又 variance

以上的假设是：人如果去分别错误几乎为 0

(optimal) base error = 0

如果 (optimal) base error = 15, 第二组结果应该就是正常的！！

判定有没有欠拟合 bias, 需要根据这个任务的 (optimal) base error !!!

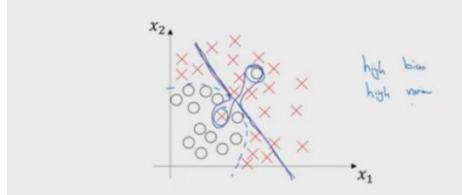
既 bias 又 variance:

在不同的 regin 会出现过拟合或欠拟合

欠拟合 整体是直线

过拟合 局部过拟合

High bias and high variance



处理流程：

1 如果有 high bias

看 train set performance, 好不好 (train set 都不好说明欠拟合, 要么 task 难!)

不好, 增加网络复杂度, 直到 train set performance 变好

2 去除 bias 以后, high variance?

如果 validate 表现不好?

1 最好的方法是 get more data 来训练模型 (减少 high variance)

2 regularization

3 合适的网络结构

直到 low bias low variance

Basic recipe for machine learning



对于深度学习：所以很少有 trade off, 不会此起彼伏, 这也是 deep learning 的魅力！

如果增加数据, 只会减少 variance, bias 不会变化, 不会增加

如果增加网络结构，只会减少 bias，variance 不会变化，不会增加
大原则：只要有正则化，增加网络结构， never hurts !!! 越大越好！只是计算量会增加。

3 Regularization (L1, L2, dropout)

Regularization: 得到一个新的 Loss function

为了 reduce variance，增加 data 成本太大，不如正则化

注意：

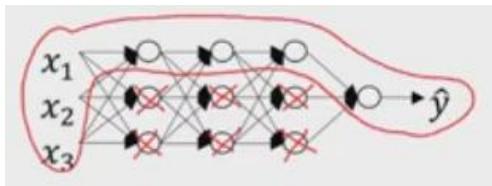
lambda 是 hyper-parameter

lambda 是 python 的保留字 lambda

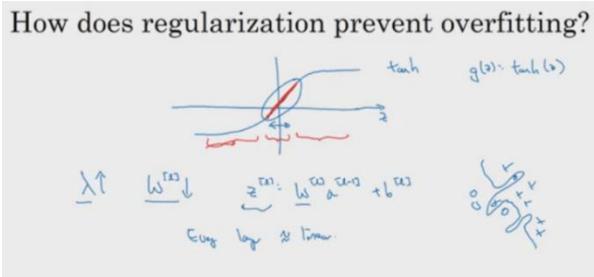
3.1 regularization 本质理解

penalize the weight too large

假设 lambda 很大，很多 w 接近 0，真正起作用的节点数不多，每一层的 hidden layer 起作用变小，成了一个简单的 simpler 网络



另一种解释



当 w 很小，z 也很小，z 集中在线性部分，激活函数还是个线性函数，因此整个网络几乎是个线性网络，很难画出非线性 boundary，很难过拟合！！！

3.2 L1 Sparse

make the model **sparse** 稀疏，部分参数为 0，可以 **compress the model** 因为部分参数为 0

Neural network

$$J(w^{(0)}, b^{(0)}, \dots, w^{(L)}, b^{(L)}) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(y_i^{(0)}, \hat{y}_i^{(0)})}_{\text{"Frobenius norm" }} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^L \|w^{(l)}\|_F^2}_{\|w^{(l)}\|_F^2 = \sum_{i=1}^{n^{(l+1)}} \sum_{j=1}^{n^{(l)}} (w_{ij}^{(l)})^2}$$

$w: (n^{(0)}, n^{(1)})$

加总所有层 L 的 w 的平方和

每一层的 w 是矩阵 (当前层节点数, 前一层节点数)

然后得到每层参数的 norm, 叫做: Frobenius norm of matrix: 矩阵元素平方和

3.3 L2 weights decay

L2 也叫: weights decay: 下面是另外一种解释
L2 更常用!!!

Logistic regression

$$\min_{w,b} J(w, b) \quad w \in \mathbb{R}^{n_x}, b \in \mathbb{R} \quad \lambda = \text{regularization parameter}$$

$$J(w, b) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(y_i^{(0)}, \hat{y}_i^{(0)})}_{\text{L2 regularization}} + \underbrace{\frac{\lambda}{2m} \|w\|_2^2}_{\text{L1 regularization}}$$

$\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

$\|w\|_1 = \frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|w\|_1$

w will be sparse

w 是个向量, 代表所有的参数, L2 就是所有参数平方和, 也就是向量的模的平方

A regularization technique (such as L2 regularization) that results in gradient descent shrinking the weights on every iteration.

对于后向传播, 参数更新

对 L2 项求导数, 得:

$$\left(\frac{d}{dW} \left(\frac{1}{2} \frac{\lambda}{m} W^2 \right) \right) = \frac{\lambda}{m} W.$$

加上后向传播的 gradient:

$$\begin{aligned} \frac{d\lambda}{dW} &= (\text{from backprop}) + \frac{\lambda}{m} w^{(0)} \\ \rightarrow w^{(0)} &:= w^{(0)} - \alpha \frac{d\lambda}{dW} \\ &\quad \text{"Weight decay"} \end{aligned}$$

$$\frac{\partial J}{\partial w^{(0)}} = \frac{\partial J}{\partial w^{(0)}}$$

$$\begin{aligned} w^{(0)} &:= w^{(0)} - \alpha \left[(\text{from backprop}) + \frac{\lambda}{m} w^{(0)} \right] \\ &= w^{(0)} - \frac{\alpha \lambda}{m} w^{(0)} - \alpha (\text{from backprop}) \end{aligned}$$

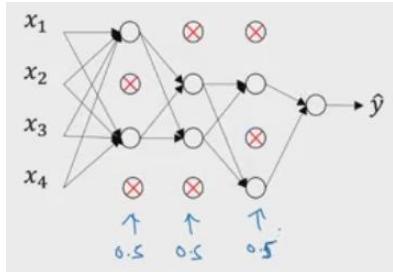
Andrew Ng

带入更新项，发现 back propa 没变，变的只是先把旧的 w 缩小 decay

andrew：use L2 , Train network as long as possible , 这样使得 hyper-parameter search 更 easy

3.4 dropout(实现)

每一层，都随机 drop 一些 nodes



每个 example, drop 的 node 不同！

用一个 reduced 网络训练，也是减少模型复杂度

Implementing dropout (“Inverted dropout”)

Illustrate with layer $l=3$. $\text{keep-prob} = 0.8$ 0.2

$$\delta^3 = \underbrace{\text{np.random.rand}(a^3.shape[0], a^3.shape[1])}_{\text{a } 3 \times 3 \text{ matrix}} < \underbrace{\text{keep-prob}}_{0.8}$$

其实就是将 drop 的节点置 0，然后再平均化

1 每一层参数，先随机产生 bool 数组 (0-1)

```
In [3]: np.random.rand(2, 3)
Out[3]:
array([[ 0.34425086,  0.7692399 ,  0.50367718],
       [ 0.56808891,  0.26411105,  0.91046691]])
```

随机产生 0-1 的数

```
In [2]: np.random.rand(2, 3)<0.7
Out[2]:
array([[ True,  True,  True],
       [ True,  True, False]], dtype=bool)
```

产生 bool 数组 mask, 0.7 是 keep_pro 也就是 drop_pro=0.3

2 然后，elementwise 相乘 $a^3 = \text{np.multiply}(a^3, \delta^3)$ ，得到该层激活值

```

In [12]: b
Out[12]:
array([[ True,  True, False],
       [False,  True,  True]], dtype=bool)

In [13]: a
Out[13]:
array([[ 0.40084936,  0.80732363,  0.64537219],
       [ 0.55720248,  0.68542644,  0.26892007]])

In [14]: np.multiply(a,b)
Out[14]:
array([[ 0.40084936,  0.80732363,  0.          ],
       [ 0.          ,  0.68542644,  0.26892007]])

```

`np.multiply(a,b)` 等于 `a*b`

3 a3/`=keep_pro` 保证每次 a3 的 expect value 一样

因为减掉了 0.3 的 unit, 下一层的 $z_4 = a_3 * w + b$ 会减少 0.3 的量
这样不好!

不同层的 dropout 要不同, 不同 example, dropout 要不同

4 test time: not use drop out

原理:

不会 put too much weight on one particular feature (使得 weight 减小)
因为任何 feature 都可能会消失

Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights. \rightsquigarrow Shrink weights.

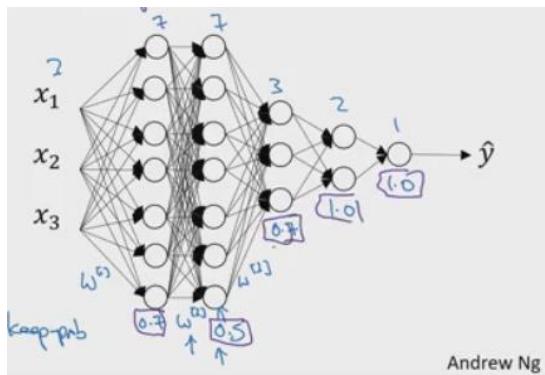


会 spread weight on 每一个 feature

使得 weight shrink, 和 L2 的效果相似

但 drop out 可以很好适应不同的结构和 weight, 不需要怎么调节

drop out 每一层也可以每一层定义不同的 `keep_pro`



Andrew Ng

对于参数较多的 layer, 担心 overfit, 可以设置小的 keep_prob

但会调整太多的 hyper-parameter

更好的选择; 某些 layer 不 drop out, 某些 drop out

drop out computer vision 应用最广！！因为 feature 太多
只有在 overfitting 情况下才用！！！

局限:

但 loss functiong 很难定义和计算, 没办法画图！！！

test 时: You do not apply dropout (do not randomly eliminate units) and do not keep the 1/keep_prob factor in the calculations used in training

4 其他 regularization 避免 overfitting 方案(代替 more data)

其他技巧 reduce overfitting: 没办法增加更多 data 时

4.1 Data Augmentation (regularzation tech)



镜像 + random_crop 随机切割

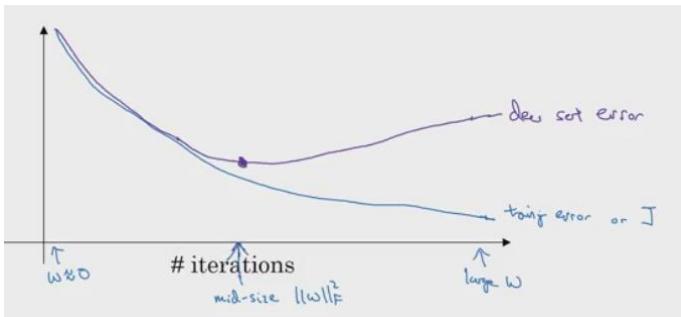


distortion 扭曲

4.2 early stopping

J 和 dev set error

随着迭代，weight 会越来越大，选择合适的大小



但 early stopping 用的比较少，因为破坏了以下原则：

既阻止了 cost function (bias) 下降，又 reduce variance，两件事缠在一起

5 训练原则：Orthogonalization

每次只考虑一个独立步骤：

因为训练模型有固定独立步骤：

- 1 先 optimize cost function (on train) (类似 gradient descent 算法)
- 2 然后 not overfit(reduce variance) (regularization, get more data)

6. Prepare for optimization problem

6.1 Normalizing inputs 标准化(归一化)+本质

标准化可以防止优化算法 stuck! ! !

机器学习都假设每个 feature 有相同的分布，如果某个特征方差太大，该特征学习到权重也大，没有意义，因此需要对特征进行归一化。

Normalizing，每个 feature 有不同的 scale，需要标准化

Normalize 是针对每一个 feature! ! ! 都要减去各自均值，再比上方差

1 subtract mean (中心化)

如下图： x 是个矩阵，包含两个 feature 向量 x_1 x_2

每个 feature 再减去其相应均值(broad casting)得到中间图

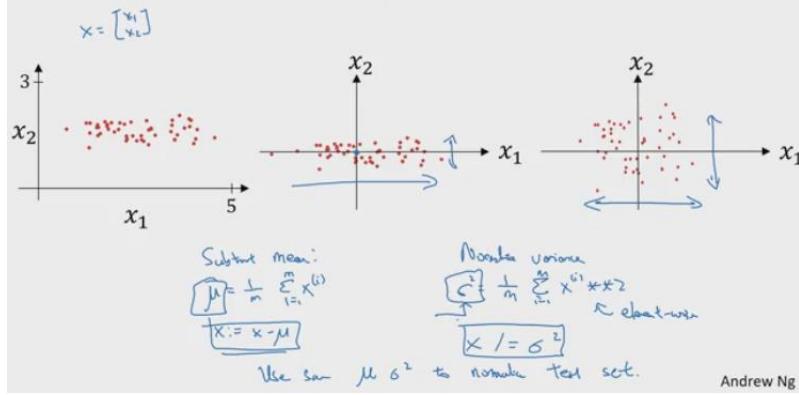
2 normalize std(标准差) 但 x_1 的方差大于 x_2

中心化数据再除以标准差

用上面的处理完的新 x (旧 $x-u$) 直接平方求均值根号，得到 std 向量：

```
np.sqrt(np.mean((X-X.mean(axis=0))**2, axis=0))
```

Normalizing training sets



然后对于 test set:

用 train 计算好的 same 均值和 std 向量，来标准化 test set!

每一列一个 feature

```
X = np.array([[ 1., -1., 2.],
              [ 2., 0., 0.],
              [ 0., 1., -1.]])
```

```
X_mean = X.mean(axis=0) #按列计算 mean
```

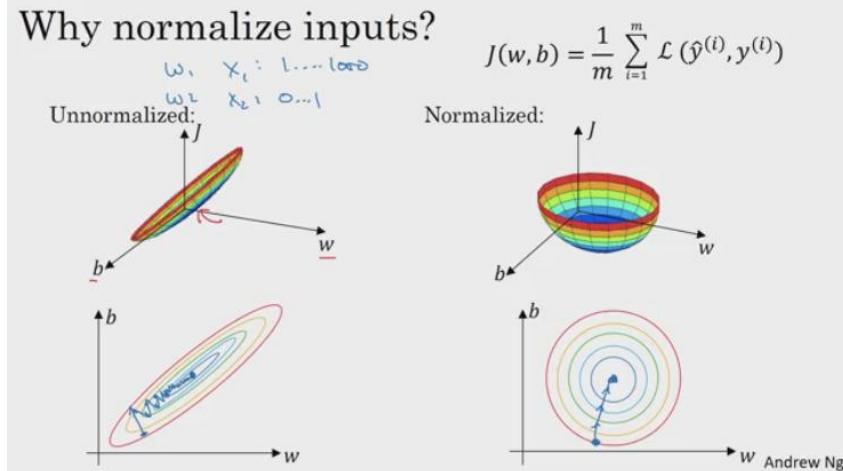
```
X_std = X.std(axis=0) #按列计算 std
```

X = (X-X_mean)/X_std

也可以直接调包:

```
X_scale = preprocessing.scale(X)
```

Why normalize inputs?

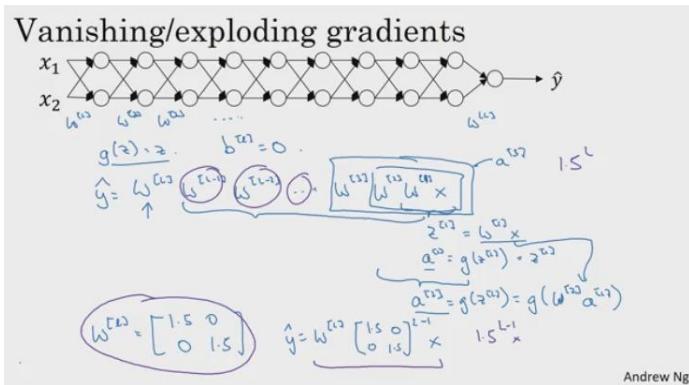


更容易优化！！

不标准化，每个 feature 的 weight 大小差别很大，会导致 cost function 很畸形！！一个维度过长 gradient descent 很慢

假如：做完 scale，通过数据的 summary 发现所有特征的数据还是集中在-1 附近，to transform these features. a log scaling might help some features. Or clipping extreme values may make the remainder of the scale more informative.

6.2 Vanishing / Exploding gradients



Andrew Ng

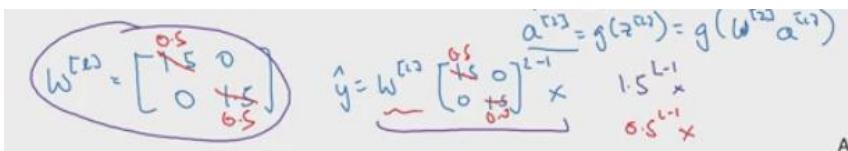
假如一个很深的 network

activation 是 linear fuc: $g(z)=z$

a 的计算就直接是 W 矩阵叠加相乘了 $WWWWWX$

假设 W 都是对角矩阵, 如果很深 1.5 的 n 次方很大, 得到的 a 值会很大, 容易 Exploding

相反, 矩阵 0.5 , a 容易 decrease!! Vanishing



而 a 是 w 的 gradient ($dw=a$), 使得参数 gradient: dw 太大或太小, 都会让参数更新有问题, 阻碍训练神经网络!!!

所以当网络越 deep, 越容易出现这种问题!!!

Batch normalization can help prevent exploding gradients, as can lowering the learning rate.

6.3 Weight Initialization for Deep Networks (not 0)

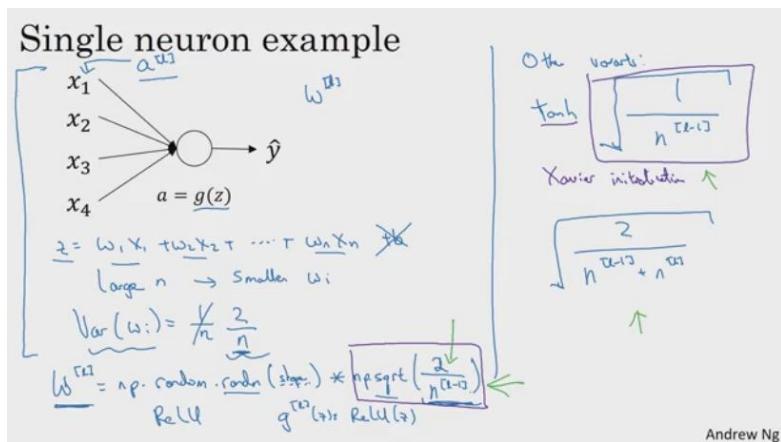
好的 weight 初始化, 可以缩短学习时间

也可以避免 gradient 爆炸和消失

symmetry: 全部初始化全部为 0 . question 合并原笔记 question

Every neuron in each layer will learn the same thing

To break symmetry, let's initialize the weights randomly



如果 input 太多，为了不让 activation 过大，需要让 weight 小一点，

1 当激活函数 relu，每一层初始化：假如这层有 n 个节点，让 weight 的方差为 $2/n$ （标准正态分布*标准差，得到数据方差为 $2/n$ ） $[0, 1]$ 变成 $[0, 2/n]$ ，使得整体 weight 变小了

每一层的 n 不同！

2 如果 tanh，可以尝试上图右边两种 var，也叫 Xavier Initialization

Finally, try "He Initialization"; this is named for the first author of He et al., 2015. (If you have heard of "Xavier initialization", this is similar except Xavier initialization uses a scaling factor for the weights $W^{[l]}$ of $\sqrt{1/\text{layers_dims}[l-1]}$ where He initialization would use $\sqrt{2/\text{layers_dims}[l-1]}$.)

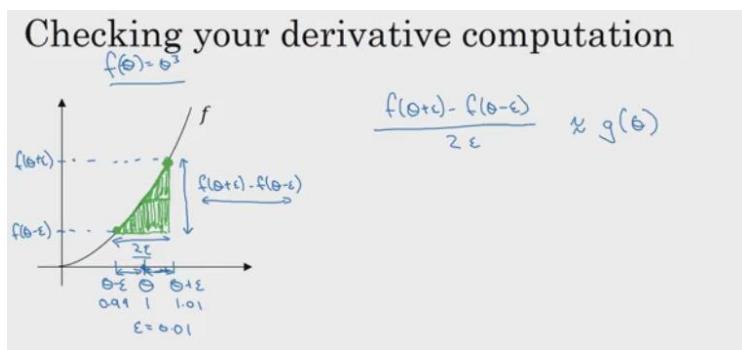
如果初始化为 0，最后一层 a 值都一样，计算的 Dj/Da 也一样， Dj/Dz 也一样
然后因为前一层 input 也一样， $dz/da[t-1]$ 也一样，导致每个参数的 gradient 都一样，同步更新！！！weight 相等，下次计算 a 还是相等

因为全局 gradient，和局部 gradient，每一层节点的公式都一样，现在值也一样，gradient 值一样！！

6.4 Numerical approximation of gradients (gradient check)

为了确保 make sure back prop right! **check the weight's gradient**

approximation of theta



两点连线的斜率，来估计 theta 的斜率，gradient

Gradient checking

Gradient check for a neural network

Take $\underbrace{W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}}$ and reshape into a big vector $\underline{\theta}$.

$$J(\underline{\theta}^{(1)}, \underline{\theta}^{(2)}, \dots, \underline{\theta}^{(L)}) = J(\underline{\theta})$$

Take $\underbrace{dW^{[1]}, db^{[1]}, \dots, dW^{[L]}, db^{[L]}}$ and reshape into a big vector $\underline{d\theta}$.

Is $d\theta$ the gradient of $J(\underline{\theta})$

将每一层的参数 reshape 成向量, 然后拼在一起成一个 big theta, 我们知道 cost function 就是 theta 的函数: $J(\text{big theta})$

以及参数的 gradient 也一样 合并成大向量: $d(\text{theta})$

看 $d(\text{theta})$ 是否是 $J(\text{big theta})$ 的导数

theta 就是参数的当前值

遍历所有 theta, 改变一点, $\theta_i + \varepsilon$ $\theta_i - \varepsilon$

求出该 theta 的 approxi gradient!! 本应该约等于 $d(\text{theta})$

Gradient checking (Grad check)

for each i :

$$\underline{d\theta_{\text{approx}}^i} = \frac{J(\underline{\theta}, \underline{\theta}, \dots, \underline{\theta}_i + \varepsilon, \dots) - J(\underline{\theta}, \underline{\theta}, \dots, \underline{\theta}_i - \varepsilon, \dots)}{2\varepsilon}$$

$\approx d\theta[i]$.

$$J(\underline{\theta}) - J(\underline{\theta}, \underline{\theta}, \dots)$$

因此我们计算两者差值

我们得到 approxi 向量和 $d(\text{theta})$ 向量

检查两个向量是否大小一致, 用向量距离, 再标准化, 除以两个向量长度平方和: 是怕向量本身太大会太小, 影响距离值, 将值标准化成一个比率

$$\begin{aligned} \text{Check } & \frac{\|d\theta_{\text{approx}} - d\theta\|_2}{\|d\theta_{\text{approx}}\|_2 + \|d\theta\|_2} \\ & \rightarrow \frac{\|d\theta_{\text{approx}} - d\theta\|_2}{\|d\theta\|_2} \\ & \varepsilon = 10^{-7} \end{aligned}$$

$\times 10^{-7} - \text{great!}$
 10^{-5}
 $\rightarrow 10^{-3} - \text{worry.}$

ε 一般选 10^{-7} 结果是 10^{-3} 就偏大了

1 如果距离太大, 需要找具体哪一个 weight i , 导致了这个差别, 遍历时候需要 track

2 J如果有 regularization, 不要忘记计算这一项!

3 需要停掉 drop out

4 可以计算多次

· Run at random initialization; perhaps again after some training.

7 Optimization algorithms

1. mini batch gradient descent

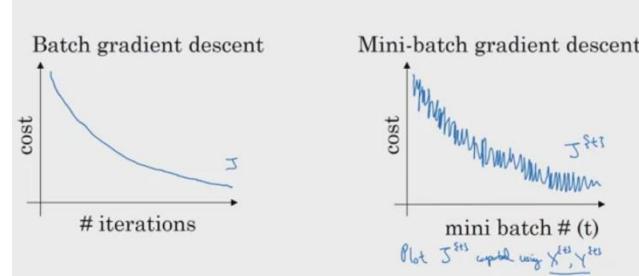
如果数据量太多, 全部数据才更新一次参数, 太慢
每 mini batch 更新一次

1 step -- 1 mini batch

用 mini batch forward then backward propagation

1 epoch: **single pass through the all training set**

Training with mini batch gradient descent



hyper-parameter: **batch size**

batch size= 1 : **stochastic gradient descent**

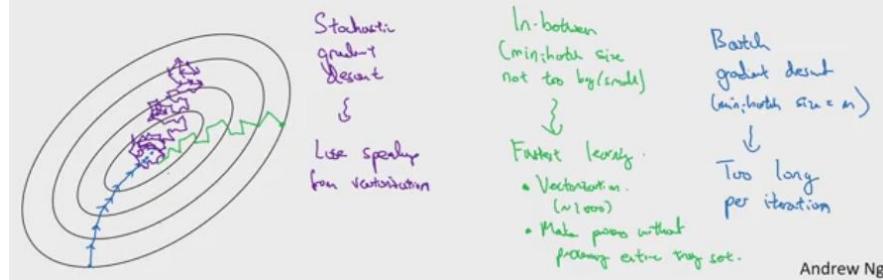
batch size= m : **batch gradient descent**

Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{(1)}, Y^{(1)}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own $(X^{(1)}, Y^{(1)}) = (x^{(1)}, y^{(1)}) \dots (x^{(n)}, y^{(n)})$ mini-batch.

In practice: Somewhere in-between 1 and m



对于 small data set use batch gradient (2000)

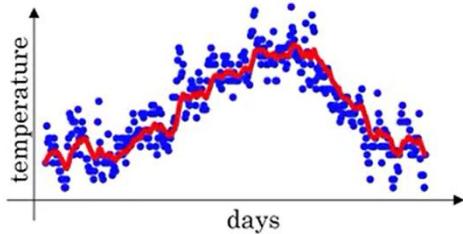
mini-batch size 64, 128, 计算更快

要保证 mini-batch size 要满足内存要求

2 更多高级 Optimal algorithm

2.1 Exponentially weighted moving averages 指数加权平滑

theta 是我们的数据，历史记录气温(紫色点)，我们想拟合！
用 V 来预测某一天的气温 $V_0=0$



假设今天 t 的维度受昨天 $t-1$ 温度影响

$$\begin{aligned}V_0 &= 0 \\V_1 &= 0.9 V_0 + 0.1 \theta_1 \\V_2 &= 0.9 V_1 + 0.1 \theta_2 \\V_3 &= 0.9 V_2 + 0.1 \theta_3 \\\vdots \\V_t &= 0.9 V_{t-1} + 0.1 \theta_t\end{aligned}$$

α 乘以昨天的温度 $(1-\alpha)$ 乘以当天的温度

$$V_t = \beta V_{t-1} + (1-\beta) \theta_t$$

$$\beta = 0.9$$

V_t : average over $1/(1-\alpha)$ 天的温度 (过去 $1/(1-\alpha)$ 天平均)

α 越大越平滑！！！

Exponentially weighted averages

$$v_t = \beta v_{t-1} + (1-\beta) \theta_t$$

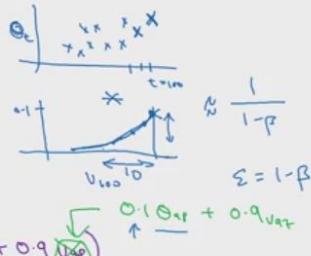
$$v_{100} = 0.9 v_{99} + 0.1 \theta_{100}$$

$$v_{99} = 0.9 v_{98} + 0.1 \theta_{99}$$

$$v_{98} = 0.9 v_{97} + 0.1 \theta_{98}$$

...

$$\begin{aligned}\rightarrow v_{100} &= 0.1 \theta_{100} + 0.9 \cancel{(0.1 \theta_{99} + 0.9 v_{99})} + 0.9 v_{99} \\&= 0.1 \theta_{100} + 0.1 \times 0.9 \cdot \theta_{99} + 0.1 (0.9)^2 \theta_{98} + 0.1 (0.9)^3 \theta_{97} + 0.1 (0.9)^4 \theta_{96} \\0.9^{10} &\approx 0.35 \approx \frac{1}{2} \quad \frac{(1-\varepsilon)^{10}}{0.9} = \frac{1}{2} \quad 0.98 ? \\&\varepsilon = 0.02 \rightarrow 0.98^{50} \approx \frac{1}{2} \quad \text{Andrew Ng}\end{aligned}$$



$$\begin{aligned}
 V_{100} &= 0.1 \underline{\theta_{100}} + 0.9 \cancel{(0.1 \underline{\theta_{99}} + 0.9 \cancel{\underline{\theta_{98}}})} \\
 &= 0.1 \underline{\theta_{100}} + 0.1 \times 0.9 \cdot \underline{\theta_{99}} + 0.1 (0.9)^2 \underline{\theta_{98}} + 0.1 (0.9)^3 \underline{\theta_{97}} + 0.1 (0.9)^4 \underline{\theta_{96}}
 \end{aligned}$$

上面是个递归式子，带入，得到每一天的 theta 乘一个系数，是一个加权平均值（第 $1/(1-\alpha)$ 天之前的系数可以忽略不计了），越近的日子系数越大，系数呈指数级减小的过程，系数之和接近 1

当指数为 $1/(1-\alpha)$ 时，系数值约等于 $1/e$ ，大于 $1/(1-\alpha)$ 的天数会减少非常快，因此，可以当做 $1/(1-\alpha)$ 天的平均结果！

得到 $V_1 V_2 V_3 \dots$ 画出图就是平滑线

implement:

$V_0 = 0$ 然后遍历 $i=1, 2, 3 \dots$ keep overwrite V

```

→ V₀ = 0
Repeat {
    Get next θᵢ
    V₀ := β V₀ + (1-β)θᵢ ←
}
Andrew Ng

```

每次获得当前的 theta，更新今天温度

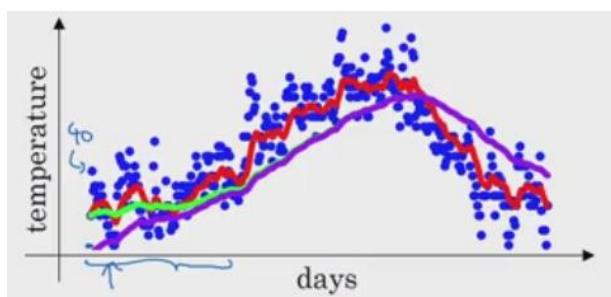
占用很小的空间，效率很高，每次只保留当前变量 V

ML 里应用很广！！！

但并不是很精确的对前 n 天进行平均！！

但比 rolling mean 更好（需要保留 n 天的内存，计算量更大）

2.2 Bias correction in exponentially weighted averages



我们想要绿色线，但实际模型成了紫色线

因为 $V_0=0$ ，前几天的温度会非常低，比正常值低，偏离了当天正常值 θ_t ，模型出现 bias！！

需要修正：

$$\frac{V_t}{1-\beta^t}$$

$$t=2: 1-\beta^t = 1-(0.98)^2 = 0.0396$$

$$\frac{V_t}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$

Andrew Ng

对当天温度进行修正，需要再除以一个算子

这样前几天的温度变大了，

后几天 t 很大时，beta 的 t 次方接近 0， V_t 几乎不变，此时得到绿色的线与紫色线重合！！！

数学之美！！！

2.3 Momentum

compute an exponentially weighted average gradient 来更新参数

Gradient descent example



用 mini-batch，会产生波动，假如是个二维的碗型曲面，总体的路径有纵向和横向，纵向的波动(oscillation)太大，会产生一个矛盾：1 slowdown gd，2 但又不能用大的 lr，因为太大波动更大 overshoot

减少纵向波动，就可以用大的 lr 收敛速度更快！！！

解决：momentum term 就如同加速度

计算 dw db 的平滑，再进行更新



纵坐标平滑后，变成 0。水平方向不变

让 gradient descent 算法 take straightforward path

滚球 ball 理解：

速度 加速度 (momentum term)

$$V_{dw} = \beta V_{dw} + (1-\beta) \frac{dw}{dt}$$

$$V_{db} = \beta V_{db} + (1-\beta) \frac{db}{dt}$$

\uparrow velocity \uparrow acceleration

beta 是用来控制速度，不要过度增长

Implementation details

On iteration t :

$$\begin{aligned} \text{Compute } dW, db \text{ on the current mini-batch} \\ v_{dW} &= \beta v_{dW} + (1 - \beta)dW \\ v_{db} &= \beta v_{db} + (1 - \beta)db \\ W &= W - \alpha v_{dW}, \quad b = b - \alpha v_{db} \end{aligned}$$

Hyperparameters: $\alpha, \beta \quad \beta = 0.9$

两个超参数 $\beta = 0.9$ (0.9 是很 robust 值) **average 10 iterations gradient**
计算平均 gradient 不需要修正

dw db 初始值是 0

也不需要 bias 纠正！！！头几步更新少一点无所谓
减少了纵向 oscillation, 可以用更大的 learning rate！！！

另外一种形式, 但不直观! 不好理解
有时 $(1-\beta)$ 会省略

$$\underline{v_{dW}} = \underbrace{\beta v_{dW} +}_{\text{dW}} \underline{dW}$$

然后下面的 learning rate 会除以一个值

2.4 RMSprop Root mean square prop

跟 momentum 一样也是为了:

- 1 为了减少梯度(mini-batch)下降中的 oscillation
- 2 可以用大的 learning rate speed up gradient descent

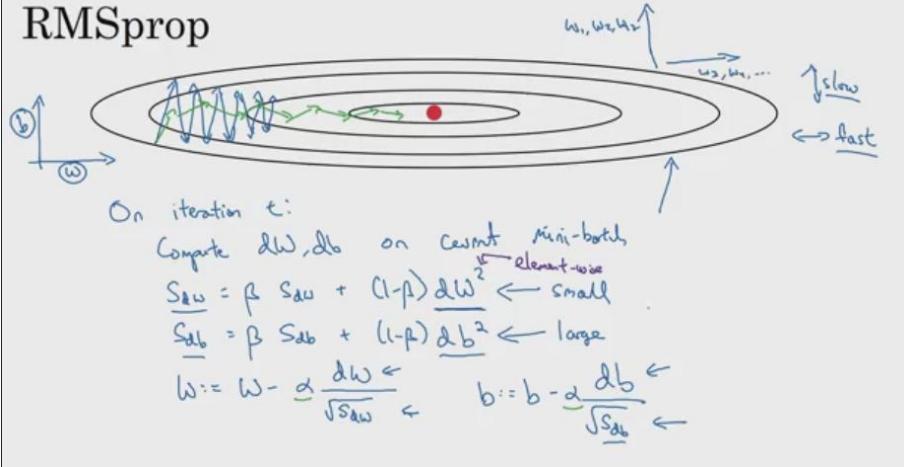
假如水平波动的参数是 w , 垂直波动的是 b

一般来讲, 每一步水平方向波动 < 垂直方向波动

因为碗型 loss 函数, 一般情况下水平方向距离小, 垂直方向距离大, 因此 db 会大于 dw , 我们想减小 db , 增大 dw

使得某一个方向的 gradient 不能过大(尤其是垂直方向值大的可能性大), 因此我们做一个 normalization!

RMSprop



1 首先计算 square of derivative dw 的平滑值(如果 dw 是向量，就元素级平分)

2 所以更新参数时用 dw 除以 S_{dw} 根号，相当于做了标准化，使得每个 dw 都不至于过大

减少纵向波动！！！可以用大的 lr 收敛速度更快

要保证根号的值不能接近 0，需要防止很小很小的数，一般需要加一个 epsilon 在分母上

2.5 Adam optimization algorithm(momentum+rms)

Adaptive moment estimation(Adam)

rms 和 adam 优化算法可以很好的适应很多深度学习结构

Adam optimization algorithm

$$V_{dw} = 0, S_{dw} = 0, V_{db} = 0, S_{db} = 0$$

On iteration t :

Compute $\frac{dW}{dw}, \frac{db}{db}$ using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) \frac{dW}{dw}, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) \frac{db}{db} \leftarrow \text{"moment"} \beta_1$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) \frac{dW^2}{dw}, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) \frac{db^2}{db} \leftarrow \text{"RMSprop"} \beta_2$$

$$V_{dw}^{correct} = V_{dw} / (1 - \beta_1^t), \quad V_{db}^{correct} = V_{db} / (1 - \beta_1^t)$$

$$S_{dw}^{correct} = S_{dw} / (1 - \beta_2^t), \quad S_{db}^{correct} = S_{db} / (1 - \beta_2^t)$$

$$w := w - \alpha \frac{V_{dw}^{correct}}{\sqrt{S_{dw}^{correct}} + \epsilon} \quad b := b - \alpha \frac{V_{db}^{correct}}{\sqrt{S_{db}^{correct}} + \epsilon}$$

Andrew Ng

先计算 momentum V_{dw} 和 rms S_{dw} ，再分别计算其修正 $V_{correct}$ 和 $S_{correct}$ ，两个参数不同 β_1 β_2

最后更新参数用修正的 momentum 比上 rms 根号，分母加上 epsilon

Hyperparameters choice:

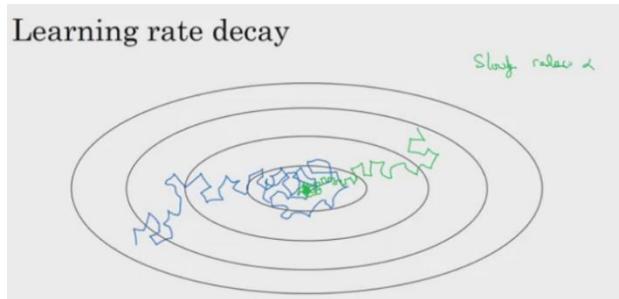
- λ : needs to be tune
- β_1 : 0.9 (du)
- β_2 : 0.999 ($d\omega^2$)
- ϵ : 10^{-8}

一般情况下用默认推荐参数，只去调 learning rate

2.6 Learning rate decay

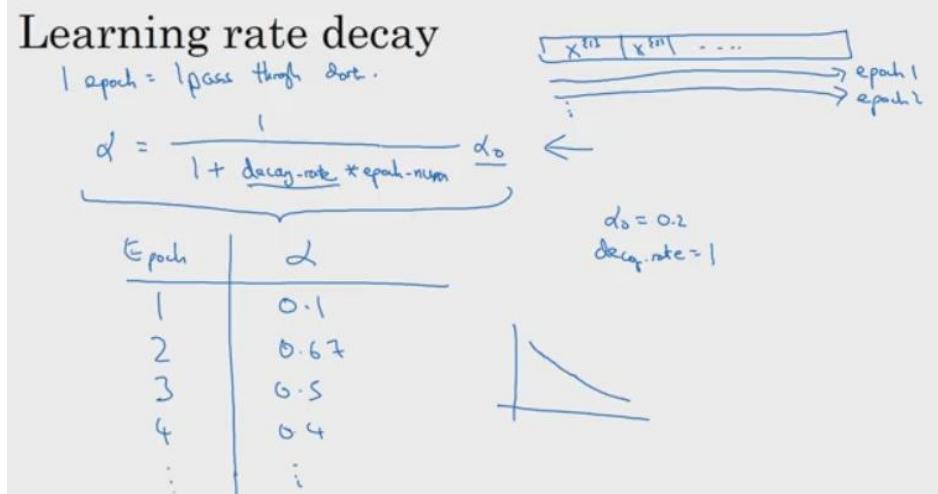
如果用固定的 learning rate，每一步都很大，noise 比较多，尤其是在靠近最优区域时，也会波动很厉害，收敛速度更慢

如果 decay，会在接近最优区域时，波动小，接近最优



1 epoch = 1 pass through the all train data

$lr = (1/\text{decay rate} * \text{epoch}) lr_0$ lr_0 是初始化 lr 值，分母递增，LR 减小



假设 $lr_0=0.2$ decay rate=1 两个超参数！！！

exponential decay

Other learning rate decay methods

forada

$$\left\{ \begin{array}{l} \alpha = 0.95^{\text{epoch_num}} \cdot \alpha_0 \quad - \text{exponentially decay} \\ \alpha = \frac{k}{\sqrt{\text{epoch_num}}} \cdot \alpha_0 \quad \text{or } \frac{k}{\sqrt{t}} \cdot \alpha_0 \end{array} \right.$$

Manual decay

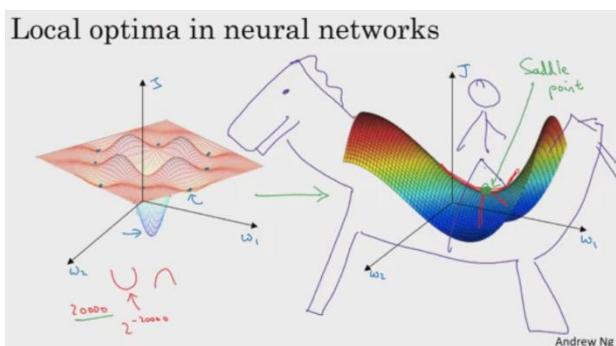
t 是 mini batch num k 是超参数 根号是 epoch-num
第三个是每次手工降一半

2.7 The problem of local optima

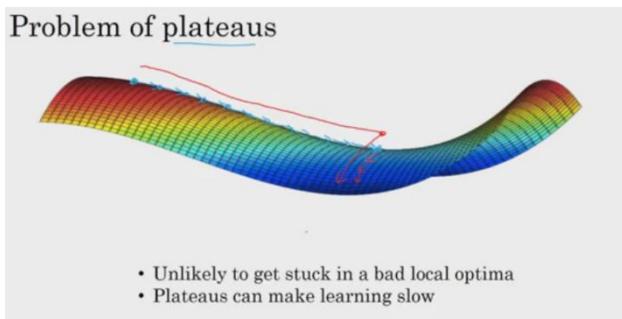
神经网络 是在高维度空间寻求最优化！！

在神经网络(高维度函数)里面很多时候, 大多是 $\text{gradient}=0$ 的 point 不是 local optimal, 而是马鞍形 saddle point(所以高维度神经网络很少会卡在 local optimal)

因为在高纬度里, 很少有凹函数, 几乎都是凸函数



saddle point 左右是弧线, 前后是峰



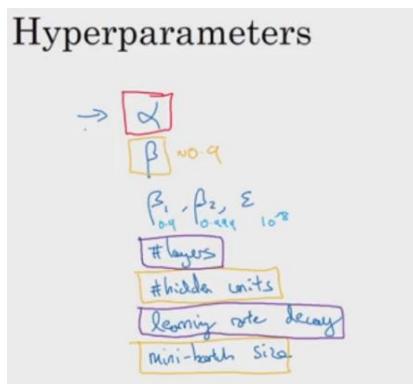
plateaus 的 gradient 接近 0(很长时间通过这个平面), 因此会使得 learn slow, 几乎不更新

因此 momentum Rms 算法会更有效, 加速学习, 越过 plateaus!! 因为 dw 不会

下降的那么快

8 Hyperparameter tuning

Tuning process



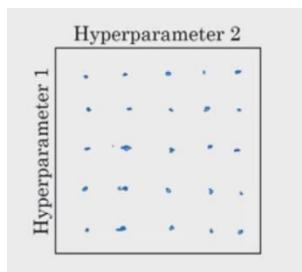
参数重要性：红>黄>紫

第一个 beta 是 momentum

第二个是 Adam 参数 一般不用变

how to search?

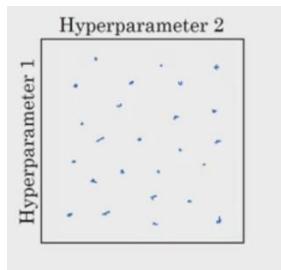
try **random** search values 不能用 grid search



因为如果参数 1 是 lr，参数 2 不重要，是 epsilon，

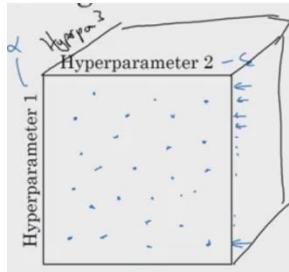
每一行都是试 5 个值，epsilon 的 5 个结果差别不大，浪费时间！！

应该是：random sample



可以尽量的多试 lr 的值。

如果三个参数：

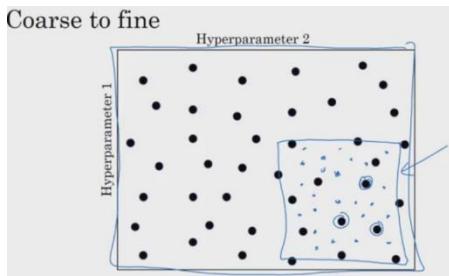


三维空间 参数空间

Coarse to fine 提炼参数

- 1 先 sample 稀疏的然后锁定一个小区域(准确率最高的区域)
- 2 然后重新 random sample(稠密的)

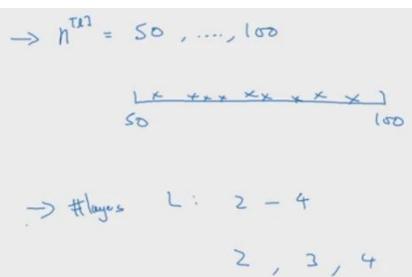
Coarse to fine



Using an appropriate scale to pick hyperparameters

如何 random 需要选择正确的 appropriate scale

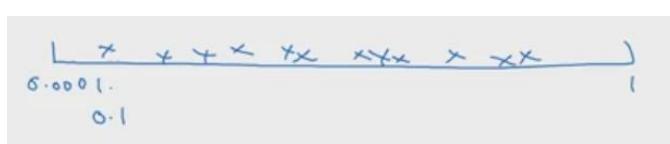
- 1 第一类参数: n 节点数和层数#layers



离散的数字，可以用一个区间 uniform sample!!

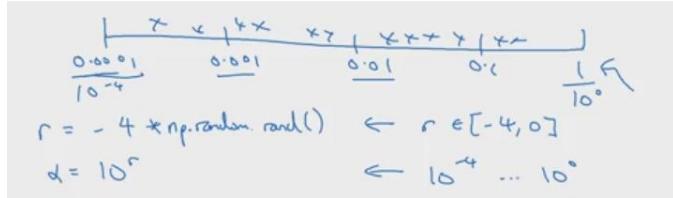
- 2 第二类: 指数 均匀分布

从 0.0001 到 1 区间长度是 0.999



如果 uniform 取值(也就是区间长度), 0.1-1 占 90%, 0.0001-0.1 占 10%
 而其实 0.0001-0.1 跨度很大(比 0.1-1 打多了)应该选择的可能性更高一点！！！
 因为每个区间的增长级别不同, linear uniform 不合理

而我们希望: 应该用 log scale, 也就是 log 值是线性均匀的
 取完 log 值以后再 linear 分布



取完 log 是: -4 -3 -2 -1

不是根据区间长度来取, 这样可以尝试更多的值, 而不只是 0.1-1 的值！！！

如: 0.0001-0.001 r[-4,-3] alpha=10⁻⁴ - 10⁻³ 0.001-0.01 r[-3,-2] 区间长度一样, 取样概率一样

log base 10 of 0.0001=-4

log base 10 of 1=0

指数 均匀分布

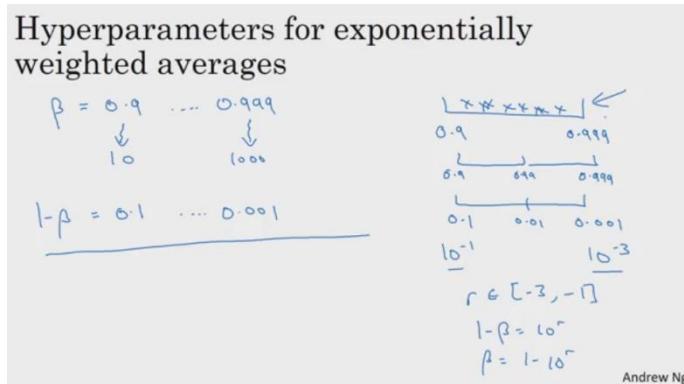
$$10^a \dots 10^b \quad r \in [a, b]$$

np.random.rand() uniform distribution over [0, 1]

利用函数变换到[a, b]区间(是连续值！！)

Adm 算法的 exponential weight average beta

例如: 0.9-0.999, 也不适合线性 sample



我们看 1-beta 刚好是我们前面的问题, 得到随机数后, 再用 1 减去就好

why do this?

当 beta 接近 1 时, sensitive 比较高, 稍微变化就会影响比较多! 所以直接在这个区间, 取均匀分布是不公平的, 增长的量级不同, 这一区域应该多取

$$\begin{aligned} \beta: & 0.900 \rightarrow 0.9005 \quad \} \sim 10 \\ \beta: & 0.999 \rightarrow 0.9995 \\ & \sim 1000 \quad \sim 2000 \quad \frac{1}{1-\beta_K} \end{aligned}$$

以上算法是让 0.99-0.999 区间采集更多的点！！！因为增长的量级更多

Hyperparameters tuning in practice Pandas vs. Caviar

如果一些模型算法改变，或配置环境改变 the set of hyperparameter get stale(过时了)

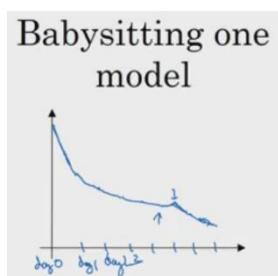
需要重新寻找最佳组合

根据 computational resources !!!

1 Babysitting one model (pandas approach) 熊猫孩子少

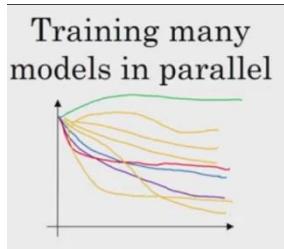
large dataset, low computational resources to train, 数据量太大，计算资源短缺

只考虑 a single model, 可以分多次训练！！每一天训练评估一次(记录当前参数，第二天从该参数开始再训练)，改进一次超参数，直到 lr 太大了，cost function 开始上升



2 Train many models in parallel (caviar approach 很多鱼卵)

能够很快的训练 model，同时尝试不同超参数的 model，越多越好，选择一个最佳



9 Batch Normalization

Normalizing activations in a network

使我们的参数初始化更加健壮，适应更多的参数值
Normalizing inputs 我们可以使得，参数 scale



更容易 gradient descent 训练

但对于多层神经网络，中间很多 activation

BN 也就是对每一隐层的 activation 进行标准化，会使得训练更有效！！

但我们实际 normalize 是 z , before activation

normalize z of layer 2, to train w , b of layer3

假设 1 个隐层， Z 是一个矩阵，类似于 X

$z(1) z(2) \dots z(m)$ 每个 example 一个 z 向量，每个向量 k 个 feature (node)
 z 向量相加，等于加总了所有 m 的每个 feature 的值，再除以 m ，得到每个 feature 的均值向量 u

$$\begin{aligned}\mu &= \frac{1}{m} \sum z^{(i)} \\ \sigma^2 &= \frac{1}{m} \sum (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}\end{aligned}$$

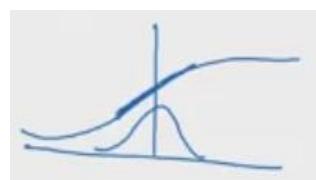
然后算出方差向量-->标准差向量

然后 $z(i)$ 向量，减去均值向量比上标准差向量，得到 $z_{\text{norm}}(i)$

需要在分母上加一个 epsilon，保证数值稳定性

此时每个 feature (node) 都是 mean 0, std 1

但我们不想每一层都是 mean 0, std 1,



标准值正态分布时 (0-1), sigmoid, 值只集中在中间线性部分！不能充分利用激活函数。

所以我们最终计算的是：每个节点多了两个可学习的参数，每个节点自己学习自

己的均值和方差

$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta \quad \text{learnable param.}$$

`Z(i) tilde=gama * znorm(i)+beta`

gama beta 是 learnable 参数

也就是计算出了 znorm 后还要，加上这两个参数，得到新的值 ztilde

当 beta=-u 时， gama=如下时

$$\begin{aligned}\gamma &= \sqrt{\sigma^2 + \epsilon} \\ \beta &= \mu\end{aligned}$$

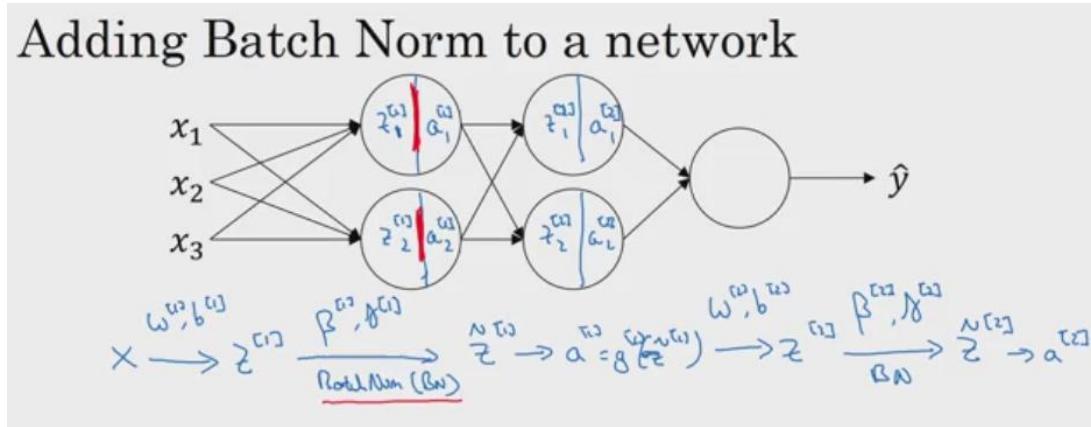
$$\begin{aligned}\mu &= \frac{1}{m} \sum z^{(i)} \\ \sigma^2 &= \frac{1}{m} \sum (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}\end{aligned}$$

ztilde 就是 z(i) 原来的值

其实就是：

`znorm` 现在是 `mean=0 std=1` 增加参数是，变成 `std` 变成 `gama`, `mean` 是 `beta`
最后用 Z tilde 作为该层最终的 z

Fitting Batch Norm into a neural network



$z^{[i]}$ 是第 i 层的 z 每一层都需要学习 $\gamma^{[i]}$ $\beta^{[i]}$ ，在激活前 normalize z

需要学习的参数：也是用优化算法更新

Parameters: $\underbrace{w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(L)}, b^{(L)}}_{\rightarrow \beta^{(1)}, \gamma^{(1)}, \beta^{(2)}, \gamma^{(2)}, \dots, \beta^{(L)}, \gamma^{(L)}} \quad d\beta^{(l)} \quad \tilde{\beta} = \beta - \alpha d\beta^{(l)}$
 $\text{tf.nn.batch_normalization} \leftarrow$

实际上，我们用 mini-batch
 每次计算 \tilde{Z} ，都只用当前的 mini-batch，独立的进行

需要注意：计算每一层 \tilde{Z} 时

Parameters: $w^{(l)}, \cancel{b^{(l)}}, \beta^{(l)}, \gamma^{(l)}$ $\tilde{z}^{(l)} = w^{(l)} a^{(l-1)} + \cancel{b^{(l)}}$
 $\tilde{z}^{(l)} = w^{(l)} a^{(l-1)}$
 $\tilde{z}^{(l)}_{norm} = \gamma^{(l)} \tilde{z}^{(l)}_{norm} + \cancel{\beta^{(l)}}$ Andrew Ng

Z 是矩阵 (k, m) ，每一列一个 example，每一行一个 node 值

1 3 5
 2 4 2
 3 2 1

b 是个向量的维度是 $(k, 1)$

上面每一行（一个 feature(node)），加上一个同一个常数 b ，每个 feature 再 normalize（最终还是要减去均值），加不加常数计算结果都一样，因此可以忽略 b 参数

Implementing gradient descent

for $t=1 \dots \text{numMiniBatches}$
 Compute forward pass on $X^{(t)}$.
 In each hidden layer, use BN to replace $\tilde{z}^{(l)}$ with $\hat{z}^{(l)}$.
 Use backprop to compute $d\tilde{w}^{(l)}, \cancel{d\tilde{b}^{(l)}}, d\beta^{(l)}, d\gamma^{(l)}$
 Update parameters $w^{(l)} := w^{(l)} - \alpha d\tilde{w}^{(l)}$
 $\beta^{(l)} := \beta^{(l)} - \alpha d\beta^{(l)}$
 $\gamma^{(l)} := \dots$

Works w/ momentum, RMSprop, Adam.

遍历每一个 mini batch

1 forward，每一隐层的 z 替换成 \tilde{z}

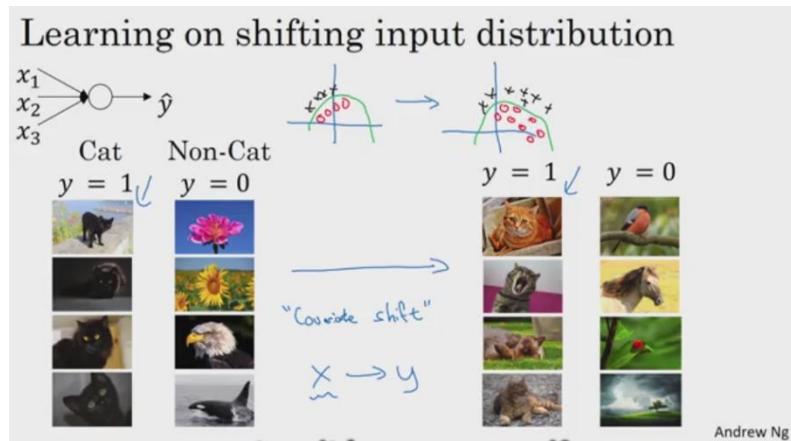
2 backforward 计算参数的 gradient， b 可以默认为 0

计算 beta gama 的 gradient!

更新参数

可以用其他优化算法

Why does Batch Norm work



第一个作用：

模型训练用黑猫， X 又变成了彩色猫，input distribution一直在变，学到的 map 也在变。

covariate shift (input 的 distribute 改变了)

当 update the parameter 时，每一层的 hidden layer values will change!
will change!

BN reduce the hidden unit values distribution shift

BN, hidden layer values 无论怎么变，都有一个稳定的 mean std

减少了 input distribution 的问题

more stable，使得下一层的参数学习更加容易

第二个作用：little regularization effect 用的少

mini batch 使得 mean 和 std 会变得 noisy(一直波动)，使得计算的 $Z_{\tilde{t}}$ 也 noisy，类似于 dropout multiply activation unit by 0 或 1 (也是 noise) BN 也是乘以 std 的倒数(BN 是乘以其他数)

not to rely on much on one hidden unit

batch size 越大，noise 越小，正则化效果越小

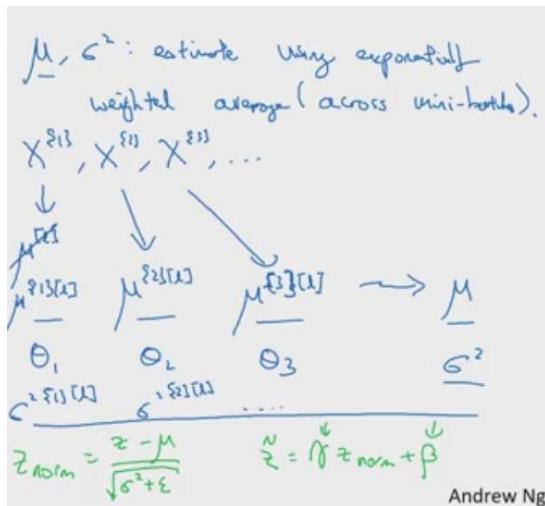
Batch Norm at test time

如果是 batch gradient descent: 全部数据

$$\begin{aligned} \mu &= \frac{1}{m} \sum_i z^{(i)} \\ \sigma^2 &= \frac{1}{m} \sum_i (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \leftarrow \\ \tilde{z}^{(i)} &= \gamma z_{\text{norm}}^{(i)} + \beta \end{aligned}$$

每一层每个节点的 u 和 std 就是全部数据的一个固定的 u 和 std
 可以直接拿来标准化 test set 的每一层的 z_{norm} , 再乘以每一层的 gama, beta
 得到每一层的 z 值

但现在是 mini-batch, 每个 batch 有个单独的 u 和 std , 一直在变
 如何 estimate u 和 std , 用 exponential weighted average, 来近似计算整个 X 的平均 u 和 std



上面的 theta 忽略！！

对于每一个 batch: $X^{(i)}$ 矩阵, 计算第 L 层的不同 mini-batch{i} 的 $\mu^{(i)[L]}$ 和 $\sigma^{(i)[L]}$
 接下来计算这些同一层的所有 minibatch 的 moving average (均值)
 然后用来计算 test 的每一层每个节点 z norm, 然后用之前 train 得来的 gama 和 beta, 得到每一层最终的 z

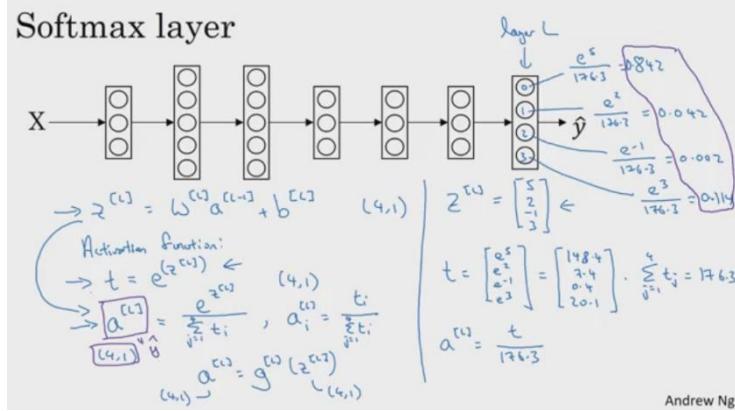
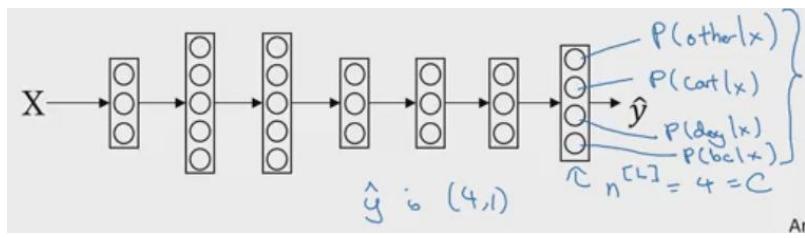
10 Multi-class classification

Softmax Regression

想分 0, 1, 2, 3 四类, 0 是其他

我们想要的结果: 每个输出节点输出的是, 每个类别的概率! 加总是 1

且输出节点从上到下的顺序和代表 01234 类别一致

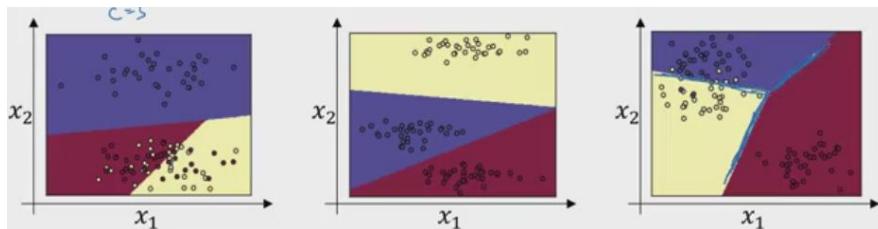


考虑最后一层 \$Z\$ 套一个特殊的激活函数 softmax

每个 \$z\$ 值作为 \$e\$ 的指数，然后归一化，除以合，输出概率

intuition:

假如没有 hidden layer, input 直接接 softmax, 会得到多个 linear boundary



可以把 softmax classifier 当做 logistics classifier 的一种扩展，扩展到多个类别！！！都是 linear boundary

但如果有了 hidden layer，就好有更加复杂的 non-linear boundary

A SOFTMAX layer generalizes SIGMOID to when there are more than two classes.

Training a softmax classifier

Understanding softmax

$(4, 1)$

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

"Soft max"

$$\alpha^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5 + e^2 + e^{-1} + e^3) \\ e^2/(e^5 + e^2 + e^{-1} + e^3) \\ e^{-1}/(e^5 + e^2 + e^{-1} + e^3) \\ e^3/(e^5 + e^2 + e^{-1} + e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

"hard max"

soft max(概率值和为 1) 和 hard max(就是 one hot encoding)

1 Cost func 定义

1 先定义 loss for a single example:

Loss function

$y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ cat $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$

$y_1 = y_3 = y_4 = 0$

$$J = -\sum_{j=1}^4 y_j \log \hat{y}_j$$

$-y_1 \log \hat{y}_2 = -\log \hat{y}_2$. Make \hat{y}_2 big.

label(y) 已经 one-hot encoding 了

loss function 定义如上: 只留下 label=1 的类别的 $-\log$ (类别概率值)

概率值越小, loss 越大! ! 越惩罚! !

因此想要 reduce loss, 就要增大正确类别的概率值(我们的目标)

loss function 的 gradient descent 寻找参数 make the corresponding class's 概率值越来越大!

2 Cost func 定义:

多个 example, 所有 loss average

$$J(\hat{w}^{(i)}, b^{(i)}, \dots) = \frac{1}{m} \sum_{i=1}^m l(\hat{y}^{(i)}, y^{(i)})$$

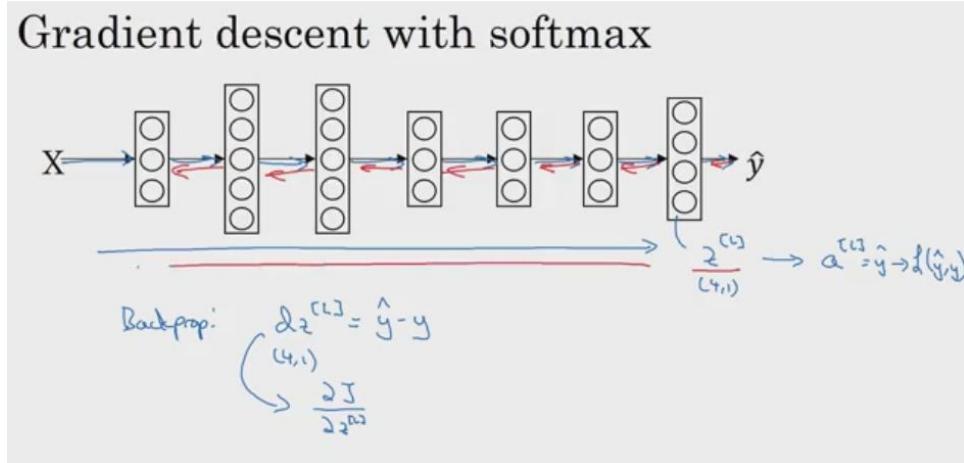
label(y) 已经 one-hot encoding 了

$$Y = \overbrace{[y^{(1)}, y^{(2)}, \dots, y^{(m)}]}^{\text{label } y} \quad \hat{Y} = \overbrace{[\hat{y}^{(1)}, \dots, \hat{y}^{(m)}]}^{\text{y hat}}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}_{(4, m)} \quad = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \quad \dots$$

label(y) 和 y hat 都是矩阵了 (k, m)

2 back propagation



核心是求 dz

$z \rightarrow a \rightarrow \text{cost}$ 先求 da , 求 dz

Backprop:

$$\begin{aligned} dz^{(L)}_{(4,1)} &= \hat{y} - y \\ &\rightarrow \frac{\partial J}{\partial z^{(L)}} \end{aligned}$$

后面的和前面的 deep learning 推导一致

11 Introduction to programming frameworks-tensorflow

- Choosing deep learning frameworks
- Ease of programming (development and deployment)
 - Running speed
 - Truly open (open source with good governance)

要预期长期的开源！！！

2. TensorFlow

$$J(w) = \frac{w^2 - 10w + 25}{(w-5)^2}$$

$w=5$

自己定义 loss func, w 取 5 时, 函数最小值!

parameter 用 tf 变量定义(一直变的)

但 tf 变量在运行前需要 init! !

变量是 Tensors (variables) that are **not yet executed/evaluated**. 需要赋值

init=tf.global_variables_initializer() 正式赋值

session.run(init) #Initializes the variables

evaluate its value

```
import numpy as np
import tensorflow as tf

w = tf.Variable(0,dtype=tf.float32)
cost = tf.add(tf.add(w**2,tf.multiply(-10.,w)),25)
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))

0.0

session.run(train)
print(session.run(w))

0.1
```

cost 就是个未执行的 tensor: 由变量和运算符组成的运算图

run (train), 只执行一次 gradient descent, 更新一次参数 w

```
for i in range(1000):
    session.run(train)
print(session.run(w))
```

run 1000 次 gradient descent, w 接近 5 了

tensorflow 已经 overload 操作符, 下面等价!!

```
#cost = tf.add(tf.add(w**2,tf.multiply(-10.,w)),25)
cost = w**2 - 10*w + 25
```

tf.constant

```
X = tf.constant(np.random.randn(3,1), name = "X")
```

```

a = tf.constant(2)
b = tf.constant(10)
c = tf.multiply(a, b)
print(c)

Tensor("Mul:0", shape=(), dtype=int32)

```

只是创建一个 Tensor, put in the 'computation graph', but you have not run this computation yet. In order to actually multiply the two numbers, you will have to **create a session and run it.**

To summarize, remember to initialize your variables, create a session and run the operations inside the session.

placeholder: 只是个占位符, 需要我们填充数据

a variable, 有

```

x = tf.placeholder(tf.float32, [3,1])
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]

```

这是 tensorflow 的一大特色思路,

因为 placeholder 的好处是, 可以处理不定的数据!! None

尤其是 mini-batch, 需要填充不同的数据

```
tf.placeholder(tf.float32, [n_x,None])
```

A placeholder is an object whose value you can specify only **later**.
tf.Varable 是立即要给赋值

A placeholder is simply a variable that you will assign data to only later, when running the session

#定义一维数组

```
tf.placeholder(tf.float32, shape=(3, ))
```

run 的时候需要填充的数据:

```
coefficients = np.array([[1.], [-10.], [25.]])
```

```
session.run(train, feed_dict={x:coefficients})
```

tensorflow 已经 bulit-in back propagation, 自动完成

Note that there are two typical ways to create and use sessions in tensorflow:

Method 1:

```

sess = tf.Session()
# Run the variables initialization (if needed), run the operations
result = sess.run(..., feed_dict = {...})
sess.close() # Close the session

```

Method 2:

```

with tf.Session() as sess:
    # run the variables initialization (if needed), run the operations
    result = sess.run(..., feed_dict = {...})
    # This takes care of closing the session for you :)

```

第二种 Session 不用我们自己 close

类型转换：

```
tf.cast(z, dtype=tf.float64)
```

或者建立一个 placeholder 指定类型，然后再传入值

```
tf.placeholder(tf.float32)
```

cross entropy 多节点？？ question

$$J = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log a^{[2](i)} + (1 - y^{(i)}) \log(1 - a^{[2](i)}))$$

```
tf.nn.sigmoid_cross_entropy_with_logits(logits = ..., labels = ...)
```

返回的是每个 example 的 cross entropy 值的数组们还需要 reduce_mean，求平均，才是我们的 loss func

input z, compute the sigmoid (to get a) and then 和 labels compute the cross entropy cost J

同时 run 多个 tensor, cost 和 optimizer 一起执行！

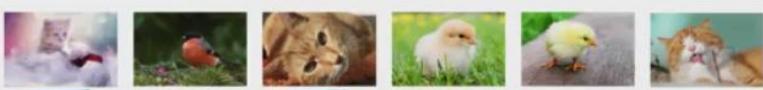
```
_ , c = sess.run([optimizer, cost], feed_dict={X: minibatch_X, Y: minibatch_Y})
```

_ 用来 throw away

Structuring Machine Learning Projects

1 大原则

Motivating example



q5%.

Ideas:

- Collect more data ↪
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add L_2 regularization
- Network architecture
- Activation functions
- # hidden units
- ...

Andrew Ng

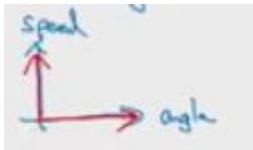
如何快速提高准确率？

DEV SET 是我们的 target，所以 dev set 一定要和最终的 test set 分布一致！！

2 Orthogonalization 垂直化

每一个按键，只完成一件事，这样比较容易调整

假设开车，有两个按钮控制方向，两个按钮控制刹车，就没啥意思了



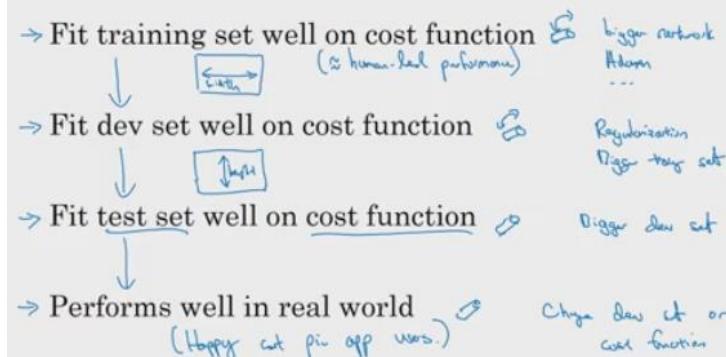
让两个功能尽量的不交叉，不要同时控制两个功能

能够更好的 tune！！！

一般不用 early stop

1 影响 train set 的表现 增加 bias 不能充分学习 2 防止过拟合提高 dev 的表现 不是正交的！！！同时影响两件事！！

Chain of assumptions in ML



每一步试图解决一个独立问题

使得 model perform well in the train set

用一个按键，来 tune，！ bigger network 或 更好的优化算法，两个独立的按键

dev 不好，train 好：用 regularization

3 Single number evaluation metric 模型评估！

主要是看模型在 dev 上面的表现！！！作为对模型的评估
来检验我们的调整的效果！！！

precision: 当分类器认为是 cat，准确率是多少
分类器预测的 cat 中，多少是真的 cat

recall: 预测对的 cat 占真正全部 cat 比例

我们想两个都高！！！

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

但如果两个值作为 metric, 很难看出哪个分类器更好！！！

用 F1 score

$$\frac{2}{\frac{1}{P} + \frac{1}{R}} \quad \text{Harmonic mean}$$

good dev set + single number metric 用一个 metric!!

很多指标的话, 很难去筛选模型！！！

Algorithm	US	China	India	Other
A	3%	7%	5%	9%
B	5%	6%	5%	10%
C	2%	3%	4%	5%
D	5%	8%	7%	2%
E	4%	5%	2%	4%
F	7%	11%	8%	12%

可以加个 average!

以上可以让分类器, 自动的选取模型

4 Satisficing + Optimizing metric(多 metric)

Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

$\text{Cost} = \underline{\text{accuracy}} - 0.5 \times \underline{\text{running time}}$

$\underline{\text{Maximize accuracy}}$
 $\underline{\text{subject to running time}} \leq \underline{100 \text{ ms}}$.

可以定义一个公式

另一种方式: 一个指标 optimizing(优化), 一个指标 satisficing(满足某个条件 threshold, 一旦不满足就不 care 这种组合情况了)

如果有 n 个 metric, pick 1 optimizing, n-1 satisficing

accuracy: $\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{FP} + \text{TN} + \text{FN})$

预测 cat, 预测正确 TP, 预测不是 cat, 预测正确 TN

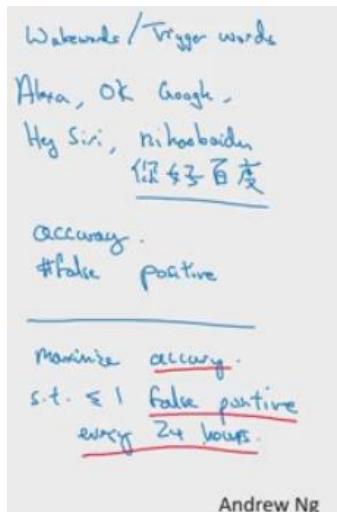
预测正确/总数

false positive: 预测 cat, 但不是

False negative

只要有 false, 就代表预测错的! 词面都是分类器预测的, 词反面是真实的!

例子: 语音唤醒设备



1 metric : accuracy 正确率

2 FP: 没有说你好, 但设备打开了

1 作为 optimizing

2 satisficing, 一天内不超过 1 次

以上可以让分类器, 自动的选取模型

5 Train/dev/test

distributions

一定保证 dev 和 test 分布相同, 因为靠 dev 调整模型性能是为了更好的 test performance ! !

假设 cat data 来自不同的国家



这样分的话, dev 和 test 来自不同的分布是不对的! !

因为这样使得 model 学习调整的方向不同, 最后 test 是另一个方向

Optimizing on dev set on loan approvals for
medium income zip codes

$\uparrow \quad x \rightarrow y$ (repay loan?)

Tested on low income zip codes

dev 用的 medium income

test 用的 low income

不同的分布，不行！！！

Guideline

Sane distribution

Choose a dev set and test set to reflect data you expect to get in the future and consider important to do well on.

首先要找到你的 **目标市场 test**，然后让 **dev** 也是这个目标市场！！！

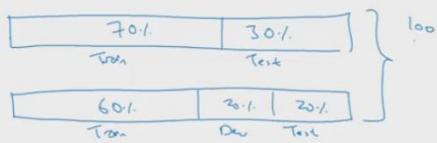
例如无人驾驶：

Your 100,000 labeled images are taken using the **front-facing camera** of your car. This is also the **distribution of data** you care most about doing well on

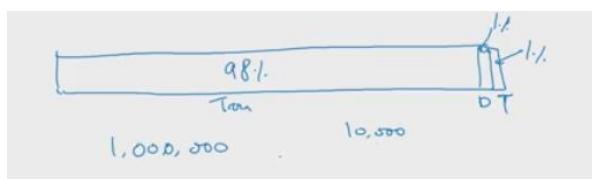
用我们手机拍的照是另一种 distribution!!

Size of the dev and test sets

Old way of splitting data



现在是 big data，不同了，比例可以小一点，但够用了！



Size of test set

→ Set your test set to be big enough to give high confidence in the overall performance of your system.

test set 只是给一个 **confidence**

也可以没有 test set, 只有 train 和 dev, 但不建议！！！

6 When to change dev test sets and evaluate metrics

pornographic 色情照片: false positive

Cat dataset examples

Metric + Dev : Prefer A
You/users : Prefer B.

Metric: classification error

Algorithm A: 3% error → pornographic

✓ Algorithm B: 5% error

A 好, 但 A 会有出现色情图片, 从 dev 和 metric 来说 A 比 B 好
但我们希望要 B, 此时有冲突, 需要 **change dev metric test**
我们用的 error:

$$\text{Error: } \frac{1}{m_{\text{dev}}} \sum_{i=1}^{m_{\text{dev}}} I\{y_{\text{pred}}^{(i)} + y^{(i)}\}$$

之前的只要 y_{pred} 不等于 y 才惩罚, 没有考虑到黄色和非黄色照片！！！

将黄色照片 label 成 cat, 我们不希望！！！

需要 define **new evaluate metric**

$$\text{Error: } \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{\text{dev}}} w^{(i)} I\{y_{\text{pred}}^{(i)} + y^{(i)}\}$$

$w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases}$

labeled value (0/1)

加一个系数, 如果 x 是 porn, 判断错就加大权重！！

unbias : 再加个归一化 error(0,1)之间, 让合等于 1

如果我们的 metric 不能给我们正确的 rank order of model give, 我们需要改进。

take machine learning problem break down **distinct step**
一个按钮 一个 step

Orthogonalization for cat pictures: anti-porn

- ⇒ 1. So far we've only discussed how to define a metric to evaluate classifiers. ← Place target ↗
 - ⇒ 2. Worry separately about how to do well on this metric. ↗
↑ Aim (shoot at target)
- $J = \frac{1}{n} \sum_{i=1}^n w^{(i)} \ell(y^{(i)}, \hat{y}^{(i)})$
- 

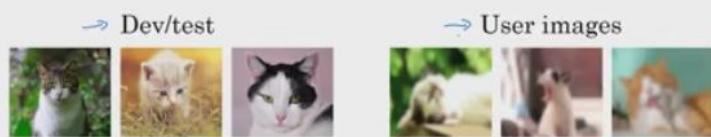
分成独立两步：

- 1 define metric
- 2 how to well on the metric

Another example

Algorithm A: 3% error

✓ Algorithm B: 5% error ↗



If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.

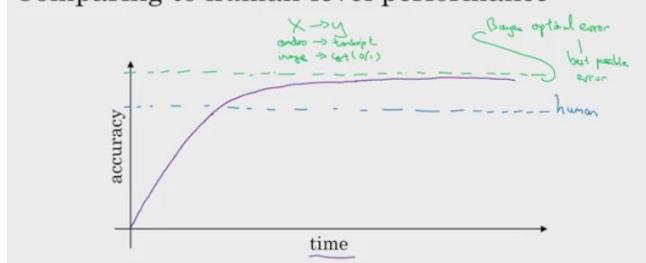
metric 和 dev 的错误率是建立在高分辨率下，但 user (test) 用的是低分辨率，模型表现差

metric 和 dev 又失败，需要改！！重新评估训练

7 bias and variance based on human-level performance

bayes optimal error 是理论的最高边界，最小的 error

Comparing to human-level performance



一旦超过人的水平就开始缓慢增速

因为人水平已经接近 bayes optimal error

机器还是先学人！！！不断的学习

Why compare to human-level performance

Humans are quite good at a lot of tasks. So long as ML is worse than humans, you can:

- Get labeled data from humans. (x, y)
- Gain insight from manual error analysis:
Why did a person get this right?
- Better analysis of bias/variance.

人可以不断的再做标记，improve model，但人的水平也有限，人来教会模型！！
但我们认为 model 的水平会超过人，因此对模型有个预期的准确率

Avoidable bias (结合 human level)

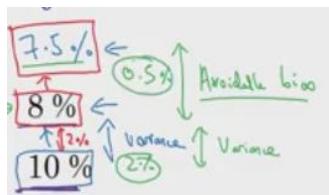
Cat classification example

Humans	1%
Training error	8%
Dev error	10%

假设：human error 是近似 bayes error

1 如果 human 和 train 有 big gap (avoidable bias=7%)，需要先提高 train 的准确率 **focus on reduce bias**: 增加网络结构，run gradient descent longer time

2 如果 human 是 7.5%，train 很好了，但 dev 不好，**过拟合**
focus on reduce variance: regularization, more train data



2%的 variance 比 0.5%的 avoid bias 更加容易

Understanding human-level performance

Human-level error as a proxy for Bayes error

Medical image classification example:



Suppose:

- (a) Typical human 3 % error
- (b) Typical doctor 1 % error
- (c) Experienced doctor 0.7 % error
- (d) Team of experienced doctors .. 0.5 % error

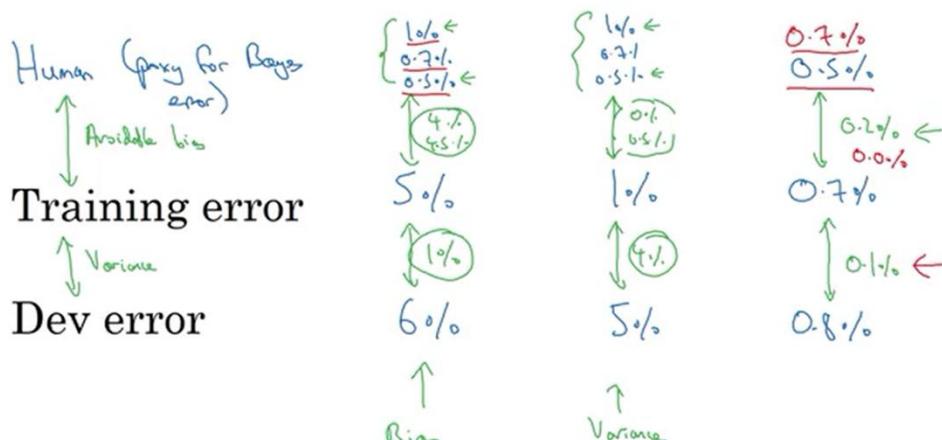
$$\text{Bayes error} \leq 0.5\%$$

What is "human-level" error?

Andrew Ng

如果是为了近似 bayes error 选错误率最低的

Error analysis example



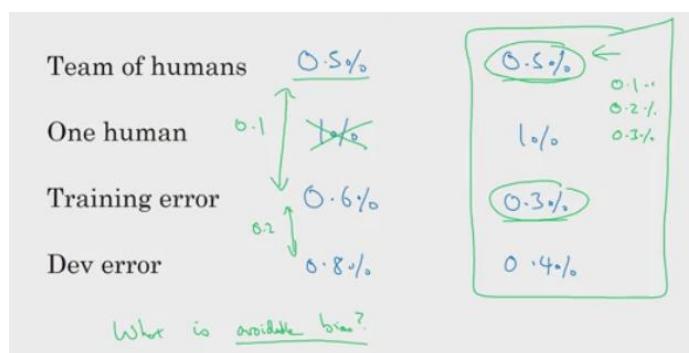
左边此时用哪个作为 human level 都无所谓, avoidable bias 都是 4 左右, avoid avarance=1 重点放在降低 bias! !

对于中间, avoidable bias 都是 0. 几左右, avoid avarance=4 重点放在降低 variance! !

右边, 需要用最低的 error, 来进一步降低 bias! !

选择合适的 human error, 可以帮我们决定是降低 bias 还是 variance

Surpassing human-level performance



左边好判断，需要选 0.5 作为 base error
但右边，一旦超过人类水平，说明 bayes 应该是更小的值~~
一旦超过了 human，很难决定下一步怎么走了

这些领域已经超过了 human:

Problems where ML significantly surpasses human-level performance

- Online advertising
- Product recommendations
- Logistics (predicting transit time)
- Loan approvals

这些不是 nature perception 问题(human 更擅长)，例如视觉，语音

Improving your model performance

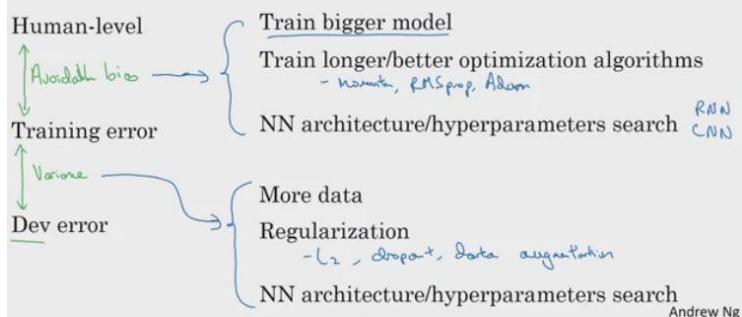
The two fundamental assumptions of supervised learning

1. You can fit the training set pretty well. 
Avoidable bias
2. The training set performance generalizes pretty well to the dev/test set. 
Variance

两个基本的假设

先看 bias，再看 variance

Reducing (avoidable) bias and variance



architecture 代表新的 activation: CNN RNN

11. You also evaluate your model on the test set, and find the following:

Human-level performance	0.1%
Training set error	2.0%
Dev set error	2.1%
Test set error	7.0%

What does this mean? (Check the two best options.)

- You should get a bigger test set.
- You have underfit to the dev set.
- You should try to get a bigger dev set.
- You have overfit to the dev set.

都存在！！！

8 error analysis

find error then improve

找到错误在哪，才能进一步指导改进方向！！

错误分析 分析 FP 和 FN

count 不同类型错误的比例 很重要！！

mis-classifying, 是几只长得像猫的狗！

Look at dev examples to evaluate ideas



→ 10% error
90% accuracy

Should you try to make your cat classifier do better on dogs? ↩

Error analysis:

- { Get ~100 mislabeled dev set examples.
- Count up how many are dogs.

"ceiling"

5% 10%
5/100 95%

50% 10%
50/100 ↓ 5%

Andrew Ng

我们是否需要更好的让模型识别狗呢？ one idea

看 dev 的错误输出

如果狗只占 5%，顶多只能改进 0.5%

如果狗只占 50%，顶多只能改进 5%，就值得做

以上是 1 个 idea，如果多个 idea

Evaluate multiple ideas in parallel

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats ←
- Fix great cats (lions, panthers, etc..) being misrecognized ←
- Improve performance on blurry images ← ↗

Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
:	:	:	:	:	
% of total	8%	43%	61%	12%	

Andrew Ng

计算每一类错误所占比例！！！

43% great cat 是 maximum amount 这种问题的解决 could improve performance.
然后挑选比例最高的问题，但也要可行

8. You decide to focus on the dev set and check by hand what are the errors due to. Here is a table summarizing your discoveries:

Overall dev set error	14.3%
Errors due to incorrectly labeled data	4.1%
Errors due to foggy pictures	8.0%
Errors due to rain drops stuck on your car's front-facing camera	2.2%
Errors due to other causes	1.0%

In this table, 4.1%, 8.0%, etc. are a fraction of the total dev set (not just examples your algorithm mislabeled). I.e. about $8.0/14.3 = 56\%$ of your errors are due to foggy pictures.

The results from this analysis implies that the team's highest priority should be to bring more foggy pictures into the training set so as to address the 8.0% of errors in that category. True/False?

- True because it is the largest category of errors. As discussed in lecture, we should prioritize the largest category of error to avoid wasting the team's time.
- True because it is greater than the other error categories added together ($8.0 > 4.1+2.2+1.0$).
- False because this would depend on how easy it is to add this data and how much you think your team thinks it'll help.
- False because data augmentation (synthesizing foggy images by clean/non-foggy images) is more efficient.

需要评估可行性！！！

data augmentation

如果某一个类别老是错误，假如自动驾驶，有雾的图片错误率高

use **data augmentation** to address foggy images. Add 1,000 pictures of fog off the internet

So long as the synthesized fog looks realistic to the human eye, you can be confident that the **synthesized data is accurately capturing the distribution of real foggy images**, since human vision is very accurate for the problem you're solving. 对于这种特定的问题，但对于其他问题不见得，比如声音 noise

Cleaning up incorrectly labeled data(robust)

train set 有些 label 标错了 (random error)

深度学习对标错的 label 比较 robust, 只要不是太多 random error, 只要 total data 很大, 不需要修改

Incorrectly labeled examples



DL algorithms are quite robust to random errors in the training set.

Systematic errors

Andrew Ng

So for example, if your labeler consistently
labels white dogs as cats,

system error: 一直把白狗标成 cat, 这样不行!

对于 dev test 标错的 label, 也要做错误分析

Error analysis

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	
Overall dev set error			10%		
Errors due to incorrect labels			0.6%		
Errors due to other causes			9.4%		

上面 6%, 不算高, 不用处理

看错误来源于 标记错的 data 的比例

左下也不需要处理, 右下边需要纠正 label, 占的比例高, 需要修改 dev label test label 也要修正

1 因为保证 dev 的结果和 test 一致, 需要保证 same distribution

Correcting incorrect dev/test set examples

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution

2 也要考虑分析分类正确的 就算是 label 错误的 data, 但是分类器分对了

Consider examining examples your algorithm got right as well as ones it got wrong. (2/)

3 修改 dev test label 可能会导致和 train 的 distribution 稍微不同！

Build your first system quickly, then iterate

training a basic model and see what mistakes it makes.

如何 focus on 这些问题

built initial model, 然后分析, error 分析, 找到问题和下一步方向

Speech recognition example



- • Noisy background
- • Café noise
- • Car noise
- • Accent
- • Far from microphone
- • Young speaker
- • Stuttering
- • ...
- • Set up dev/test set and metric
- Build initial system quickly
- Use Bias/Variance analysis & Error analysis to prioritize next steps.

Guideline:
Build your first
system quickly,
then iterate

Andrew Ng

不要太 over think! ! ! 先建模试试

Training and testing on different distributions

我们从网上搜集照片，但我们手机 app 用的低分辨率
分布不同

1 可以所有图片混合在一起，然后 random shuffle, train 和 dev, test (此时分布相同了，不好：dev 更多的是 high 分辨率(仍然不是我们的目标 target)，很少低分辨率，我们更关注低分辨率)，这个行不通！！

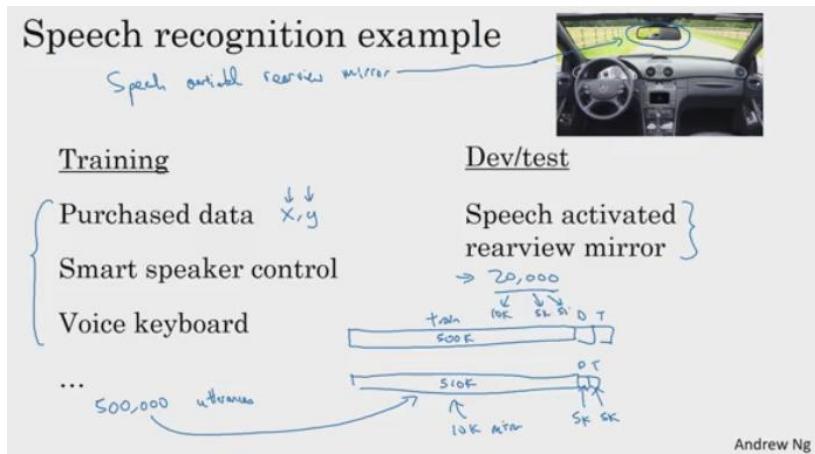
2 假设 web pic 20w, app pic 10000

将 app pic 10000 给 train, 共 205000 张 train

dev 2500 和 test 2500 全部是 app low resolution pic

明确了 target dev 是 app pic! 但唯一缺点是 train 不是同分布语音识别：

x 语音，y 是字幕



train 和 dev test 分布不同，可以分一部分给 train, dev 和 test 用我们真正关心的 data

1 Bias and Variance with mismatched data distributions

Cat classifier example

Assume humans get $\approx 0\%$ error.

Training error 1%
Dev error 10%

如果 train 和 dev 同分布，可以说是 variance 大
但不是同一分布，就不能下这个结论！！

因为 dev 是模糊照片，所以表现差一点，但并不代表有 variance，而是因为 dev 包含对于分类器比较难分类的照片

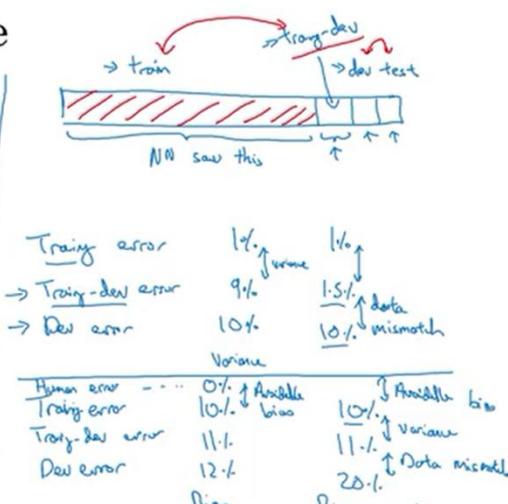
为了区分增加一个 train-dev set，是从 train 里随机划分出来，和 train 一样分布的 set，但不参与模型的 train

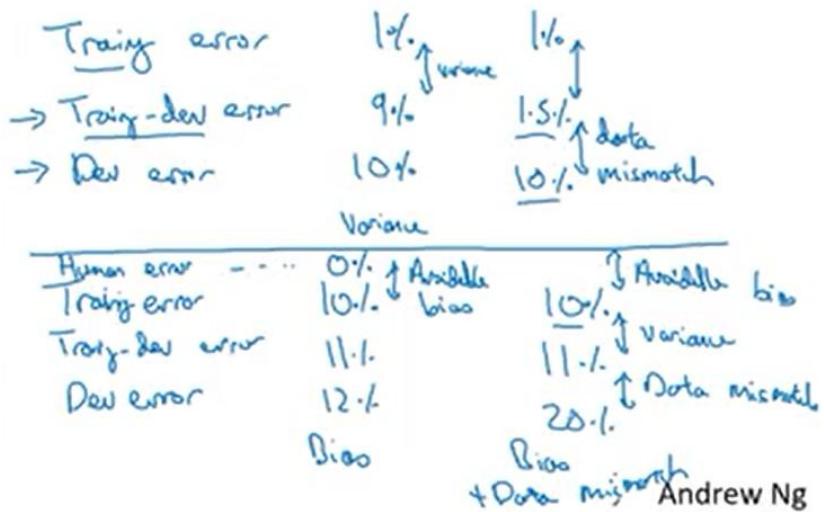
Cat classifier example

Assume humans get $\approx 0\%$ error.

Training error 1% ↓ 9%
Dev error 10%

Training-dev set: Same distribution as training set, but not used for training





如果 train error 和 train-dev error 差别大，说明是 variance 问题

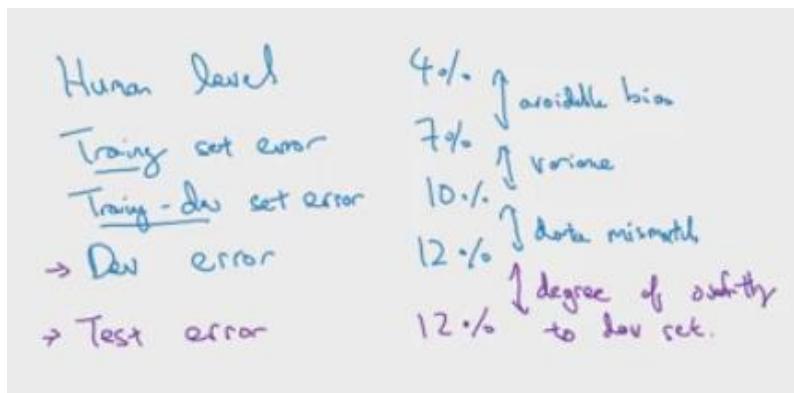
否则，并且如果 dev error 和 train-dev error 差别大说明是 data mismatch 问题：train set 和 dev set 不是一个分布

如果 train error 很大，但 human error 几乎为 0，说明 avoidable bias

如果 train error 很大，但 human error 几乎为 0， dev error 更大，说明既有 bias 又有 data mismatch

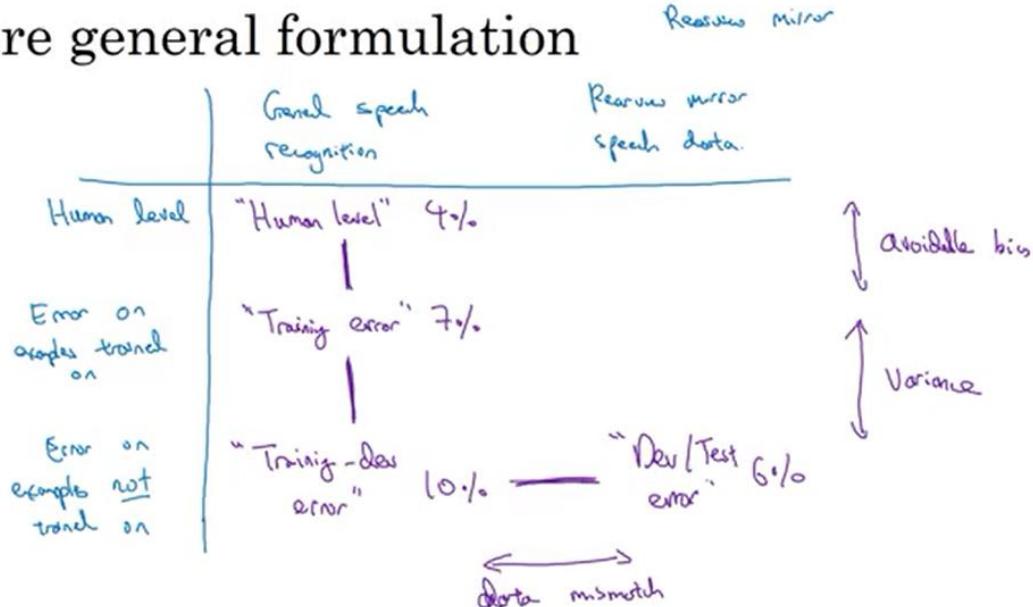
如果 dev error 和 test error 有 gap，说明在 dev 上 overfitting，需要 get more dev set data

重要的几个指标：



紫色是 dev 的 overfitting

More general formulation



左边是广泛搜集的语音 data(广泛的 distribution), 右边是我们 specific 语音 data

1 human level

2 error on example(trained on) 此时用的是广泛 data 来 train

3 error on example(not trained on) 如 train-dev 和 dev/test

上面几个指标可以告诉我们下一步的改进方向, avoidable bias, variance, data mismatch

对于剩余的空格也可以填入, 也很有作用, 更多的 insight

1 人工标注 mirror data 看准确率多少

2 用 mirror 训练, 并看 error on train 准确率多少

2 Addressing data mismatch(可以 data augmentation)

Addressing data mismatch

- Carry out manual error analysis to try to understand difference between training and dev/test sets

E.g. noisy - car noise street numbers

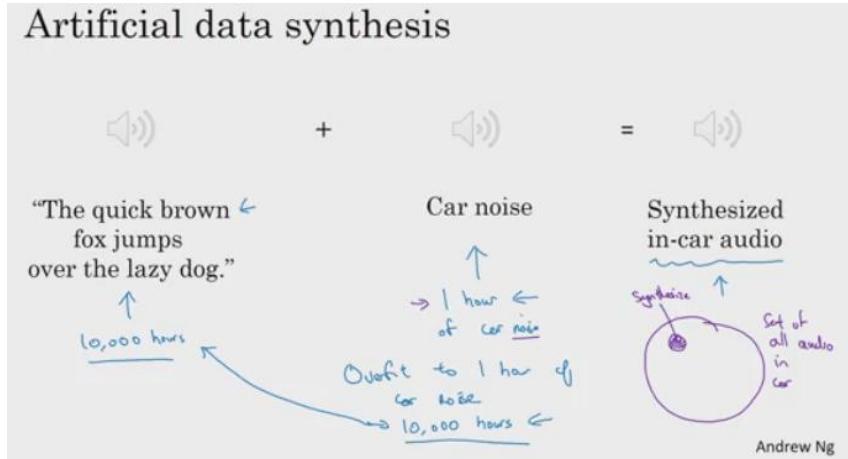
- Make training data more similar; or collect more data similar to dev/test sets

E.g. Simulate noisy in-car data

1 首先找到 **dismatch** 在何处, 例如 test 有一些背景 noise

2 然后让 train 也增加 noise, 或者搜集些和 dev test 像的数据到 train set 里, 可以人工数据合成:

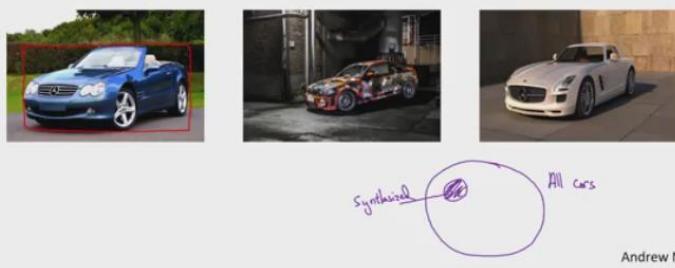
Artificial data synthesis



但如果 noise 只有 1 小时，也可能容易过拟合这一小部分的 car noise，这部分 noise 只是 dev/test noise 的一个很小的子集
和 dev/test 的分布还是不同
图片合成也是一样

Artificial data synthesis

Car recognition:

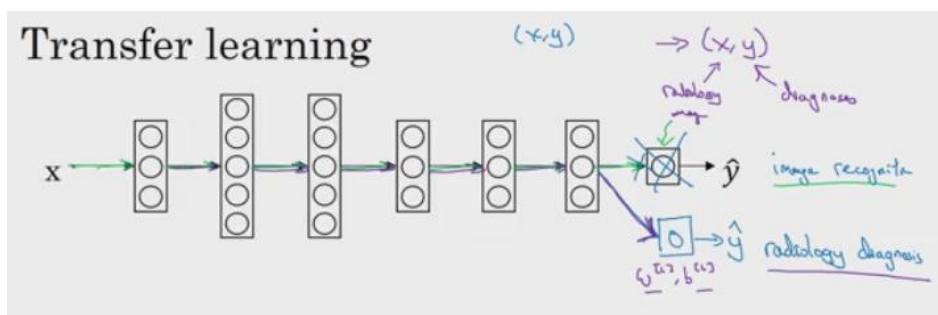


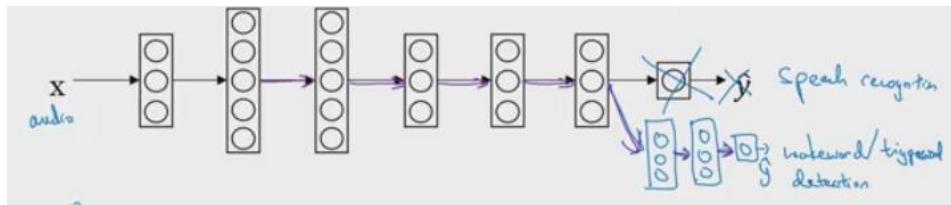
合成对于 human, 人眼, 耳朵来说, 没有什么区别, 但对于机器却有区别, 因为现实世界中的车, 千变万化, 合成永远只是子集, 容易 overfitting 子集

9 Transfer learning

很有用, 当 B Task 数据量很小时, 借助相似或相关的 A Task 的训练经验, 完成 B!

一个用途 task 的 network, 用于另一个用途 task!!





去掉原来的输出节点(最后一层)，替换为全新的输出节点(可以多个，也可以多层新的节点)，新的 weight, bias 参数，random initialize 参数。

然后 retrain 这个 network，两种方式：

1 如果新的 data X 数量很少，可以保留原来其它层的参数 keep fixed 不变，只训练最后输出层参数；

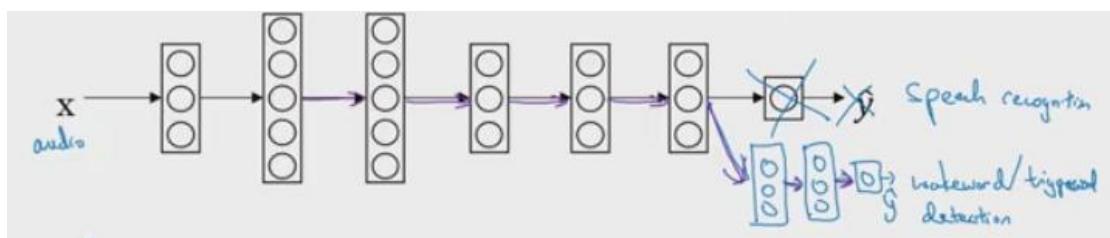
2 如果 data 足够多，可以 train 所有参数(或者更多层的参数！)，此时参数不需要再专门初始化，叫 pre-training(已经 train 了一些 weight)

然后用新的 X 来 train，叫 fine tuning

在已有的参数基础上，不需要初始化，finetune！！！

用已经训练的知识来完成其他 task，原理：

low level layer，是识别一些图形的基本组成成分，如角度，线条，对于识别其他图像 task 也是有帮助的



之前是语音识别系统，语音-文字

现在是唤醒系统，去掉原来 output，替换新的 output，retrain

Transfer learning，一般是从**大规模**数据的模型，迁移到**小数据**的应用上。如果迁移到更大的数据不合适！

When transfer learning makes sense

Train from A \rightarrow B

- Task A and B have the same input x.
- You have a lot more data for Task A than Task B.
- Low level features from A could be helpful for learning B.

1 same X image 或者 语音

2 数据量

3 A 的 low level feature 对 B 有帮助

例子：自动驾驶

So far your algorithm only recognizes **red and green** traffic lights. One of your colleagues in the startup is starting to work on recognizing a

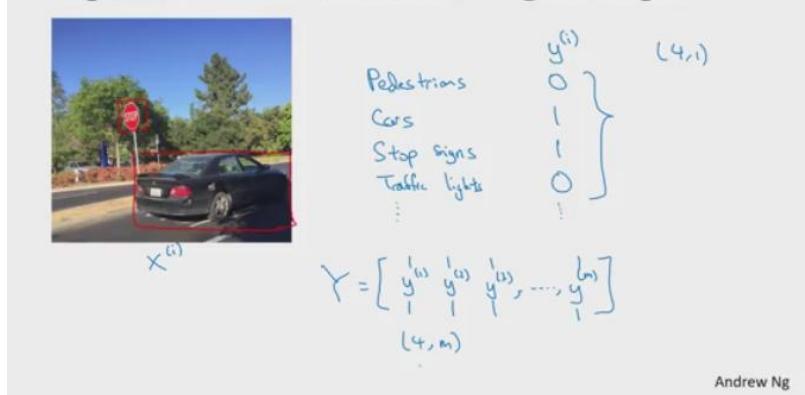
yellow traffic light. Images containing yellow lights are **quite rare**, and she doesn't have enough data to build a good model. She hopes you can help her out **using transfer learning**.

She should try using **weights pre-trained** on your dataset, and **fine-tuning** further with the yellow-light dataset.

10 Multi-task learning

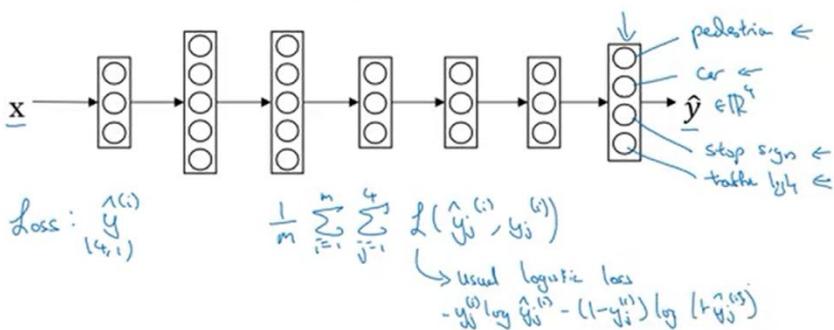
train 一个网络完成 **多个分类任务**！ 一张图片可能多个 label
会比独立的去 train separate model for every task 效果好
Object detection 自动驾驶车需要 detect 一张图多类别图片

Simplified autonomous driving example



例如图片有 stop sign 和 car, 因此 y 是个 4 维的变量
所有 example Y 就是 $4 \times m$ 维度矩阵
input 是 x 一张图片, 输出四个节点

Neural network architecture



因为每个图片可能有多个 label, 不像之前一张图片只有一个 label, 不能再用 softmax regression, 不能再用 a softmax activation, 而是每个节点都是 sigmoid(0-1)

需要重新定义 loss func , 需要 sum 每个输出节点的 logistics loss, 因为每个节点都需要单独的取考虑, 出差就要惩罚

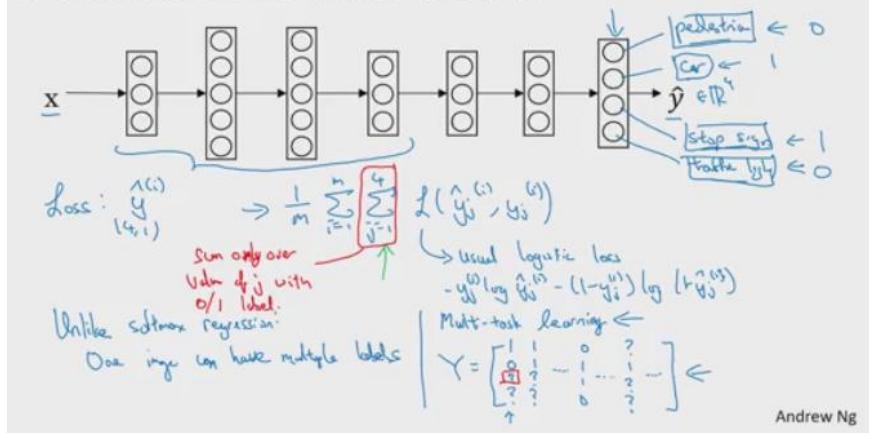
如上图: sum 4 个节点的 loss!!!

跟 softmax 不同, softmax 只考虑一个节点的 loss

(另一种解决办法就是 train 多个 network), 但如果多个 object 的 low level feature 可以被共享, 还是上面一个 network 更有效!!!!

还有个优势: 就算有些 label, 信息不全, 如 (1, 0, ?, ?)

Neural network architecture



同样还可以用!! 处理方式: 计算 loss 时, 只 sum label 是 0 和 1 的情况, 忽略? 的情况

When multi-task learning makes sense

- Training on a set of tasks that could benefit from having shared lower-level features.
 - Usually: Amount of data you have for each task is quite similar.
- | | |
|--|--|
| $A \quad \underline{1,000,000}$
\downarrow
$B \quad \underline{1,000}$ | |
|--|--|
- Can train a big enough neural network to do well on all the tasks.

1 share low level feature

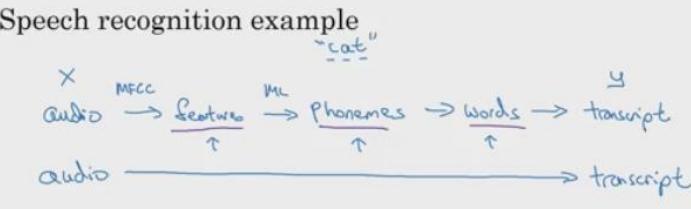
2 每个 task 的数据量相当, 假如 100 个 task, 每个 task 1000 个数据, 但我们在处理一个 task A 时, 可以借助其他 99000 数据来提供一些 knowledge 来帮助 task A(如提供一些 low level feature)

不像 transfer, 10000data \rightarrow 100 data task, 每个 task 数据量一致

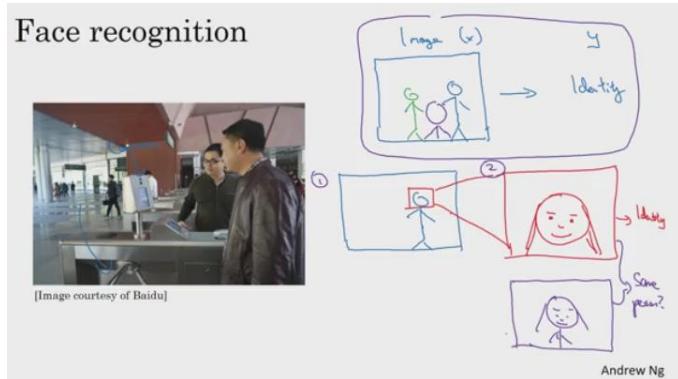
3 如果不能一次性 train a big network, 就会比 train separate 的 network of every task 效果差

11 End-to-end deep learning

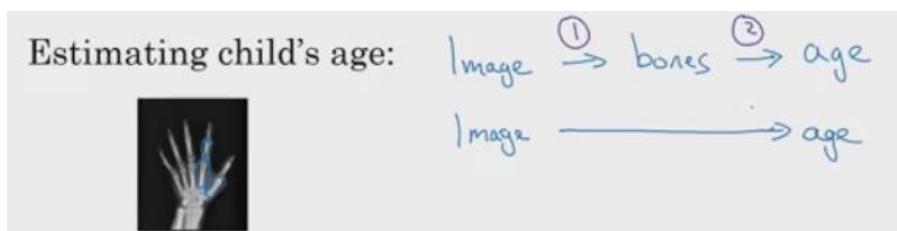
What is end-to-end learning?



将前面的 pipeline 省去，放在一个 network 里
但只有 big data 时才有明显的效果
不然还是传统的效果好
就像人脸识别一样，从人脸直接到识别结果，end to end



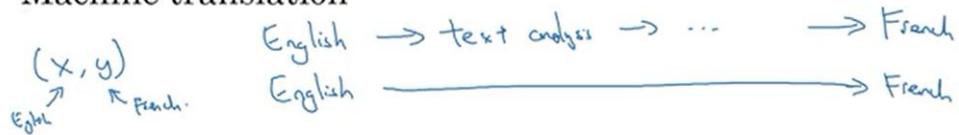
如何构建 end to end:
不是直接拿到图片就是识别
而是 1 先 detect 脸 2 再单独识别脸
我们的数据量只够单独完成一项任务，如果同时完成两个任务 end to end，数据量不够！！
不如 break down 成独立的子 sub task



直接训练 image-age end to end 需要太多的数据
不如 1 先分解骨骼 2 再根据骨骼测量 age

end to end 成功的例子：翻译系统

Machine translation



end-to-end deep learning

pros:

1 lets the data speak $X \rightarrow Y$

而不是带 human perception 人类感知，更加客观，直接去学习，不带人工处理和干预，消除了错误的主观判断

2 更方便，省去了很多手工处理过程

cons:

1 需要太多数据

2 可能错过了一些有用的手工处理，如平滑去噪，还有一些人工的指导和经验
所以关键是要衡量 $X \rightarrow Y$ 的复杂性，看数据是否够用，够用就可以 end2end

就像自动驾驶 从 image \rightarrow steer 肯定不是很适合！！

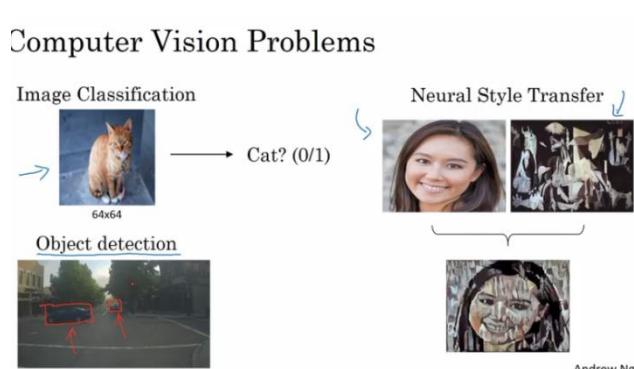
Convolutional Neural Networks (结构很重要，最好用经典结构！！才高准确

率)

此时 filter 就是每个 hidden layer tensor 的参数，不再是每个 feature 一个 weight，而是共享一套参数 (slide)，并通过后向传播，学习得到的！！！

Filter 是为了 detect feature in the data！！！免去了手动提特征的问题

1 computer-vision problem

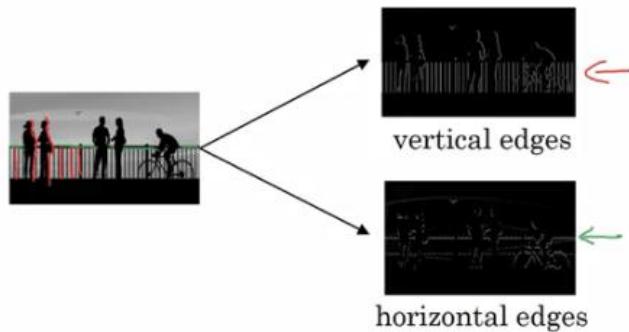


Input 太大了

2 filter 实现 feature detection

edge-detection-example

filter



横向和纵向边界

纵向边界实现：

Elementwise product, dot

Vertical edge detection

$$3x1 + 1x1 - 2x1 + 0x0 + 5x0 + 1x1 + 8x-1 + 2x-1 = -5$$

3	0	1	2	7	4
-1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6×6

"convolution"

$$\begin{matrix} & * \\ & \downarrow \\ \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = \\ \begin{matrix} -5 & & \\ & & \end{matrix} & \begin{matrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{matrix} \\ \begin{matrix} 3 \times 3 \\ \text{filter} \end{matrix} & \begin{matrix} 4 \times 4 \\ \text{result} \end{matrix} \end{matrix}$$

Vertical edge detection

$$3x1 + 1x1 - 2x1 + 0x0 + 5x0 + 1x1 + 8x-1 + 2x-1 = -5$$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6×6

"convolution"

$$\begin{matrix} & * \\ & \downarrow \\ \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = \\ \begin{matrix} -5 & -4 \\ & \end{matrix} & \begin{matrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{matrix} \\ \begin{matrix} 3 \times 3 \\ \text{filter} \end{matrix} & \begin{matrix} 4 \times 4 \\ \text{result} \end{matrix} \end{matrix}$$

Convolution operation

Vertical edge detection

$$\begin{matrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{matrix} \quad \begin{matrix} & * \\ & \downarrow \\ \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} & = \\ \begin{matrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{matrix} & \begin{matrix} & & \\ & & \\ & & \\ & & \\ & & \end{matrix} \end{matrix}$$

6×6

3×3

↑ ↑ ↑

Andrew Ng

Edge 边界在中间，右边白色的捕捉到了区域

more-edge-detection

$$\begin{array}{|c|c|c|c|c|c|} \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline 0 & -30 & -30 & 0 \\ \hline 0 & -30 & -30 & 0 \\ \hline 0 & -30 & -30 & 0 \\ \hline 0 & -30 & -30 & 0 \\ \hline \end{array}$$

Andrew Ng

filter 对不同边界图片的识别结果不同

Vertical and Horizontal Edge Detection

Vertical

Horizontal

$$\begin{array}{|c|c|c|c|c|c|} \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 \\ \hline 30 & 10 & -10 & -30 \\ \hline 30 & 10 & -10 & -30 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Andrew Ng

两种 filter, find 不同的 edge

filter 可以理解为 feature detector

另一种特殊的 vertical filter: sobel filter, 将中间一行的数变大, 大的权重(10, -10), 是的更 robust

Sobel filter

Scharr filter

而我们可以通过学习得到(反向传播), filter 的具体参数, 来完成图片的 edge detect!!! 学习到的不止是简单的水平和垂直 edge, 更复杂的 edge!

得到 edge 就是我们的特征!!!

filter 结果尺寸:

$\begin{matrix} 6 \times 6 \\ n \times n \end{matrix} \quad * \quad \begin{matrix} 3 \times 3 \\ f \times f \end{matrix} = \quad \overrightarrow{\underline{\underline{4 \times 4}}}$

固定计算公式，得到结果尺寸 $n-f+1$

3 padding(对 input image padding, 避免边缘信息丢失)

benefits of padding :

1 适当的增加 padding allows you to use a CONV layer without necessarily shrinking the height and width of the volumes(原层尺寸不会下降的太厉害, 没办法很深了 deep net, 像素下降太快了!). An important special case is the "same" convolution, in which the height/width is exactly preserved after one layer.

2 It helps us keep more of the information at the border of an image.
(使得图像边界的信息能平等的利用, 不然, 边界只扫一次, 其他像素会被扫过很多次) 不然会损失边界像素的信息

因此对于初始 input 照片进行 padding 是个好的习惯!!!

Padding

$\begin{matrix} 8 \times 8 \\ n+2p \times n+2p \end{matrix} \quad * \quad \begin{matrix} 3 \times 3 \\ f \times f \end{matrix} = \quad \begin{matrix} 6 \times 6 \\ 6 \times 6 \end{matrix}$

padding=1 padding 外边界填充一个像素!! 填充 0
得到 $n+2p-f+1$ output, 图片没有改变大小

padding 多少?

- 1 Valid padding: no padding $n-f+1$
- 2 Same Padding $n+2p-f+1=n$ $p=(f-1)/2$ p 是 padding 个数

Valid and Same convolutions

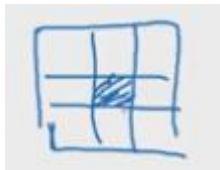
“Valid”: $n \times n$ \star $f \times f$ $\rightarrow \frac{n-f+1}{s} \times \frac{n-f+1}{s}$

$$\begin{array}{c} n \times n \\ 6 \times 6 \end{array} \star \begin{array}{c} f \times f \\ 3 \times 3 \end{array} \rightarrow \frac{6-3+1}{2} \times \frac{6-3+1}{2} = 4 \times 4$$

“Same”: Pad so that output size is the same as the input size.

$$\begin{array}{c} n+2p-f+1 \\ n+2p-f+1 = n \Rightarrow p = \frac{f-1}{2} \end{array}$$

因此 filter 一般是奇数！！！



1 会有一个 central position

2 保证 p 整数, p 是小数 $p=(f-1)/2$

4 strided convolutions(完整计算公式)

Strided convolution

2	3	7	3	4	4	6	4	2	9
6	6	9	1	8	0	7	2	4	3
3	4	8	-1	3	0	8	3	9	7
7	8	3	6	6	6	3	4		
4	2	1	8	3	4	6			
3	2	4	1	9	8	3			
0	1	3	9	2	1	4			

7×7

\star

3	4	4
1	0	2
-1	0	3

3×3

$=$

9	1	0	0

stride = 2

strides=2 得到 3×3 output

output 公式: $(n+2p-f)/s + 1 * (n+2p-f)/s + 1$

因为上下和左右 stride 可以不同, filter 宽度和高度可以不同, 照片 n 的宽度高度可以不同

如果结果不是整数, 需要做 floor 操作, down to nearest integer

也就是一旦 filter 移动超过图片边界, 就舍弃! (会损失信息, 尽量要整数!!)
而 3×3 filter. 会完全的贴合原图片!

Summary of convolutions

$n \times n$ image $f \times f$ filter

padding p stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \underbrace{\frac{n+2p-f}{s}}_{+1} + 1 \right\rfloor$$

有时会将 filter 镜像处理(cross-correlation): 再卷积操作

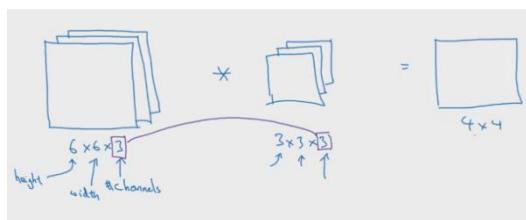
Convolution in math textbook:

对角线交换

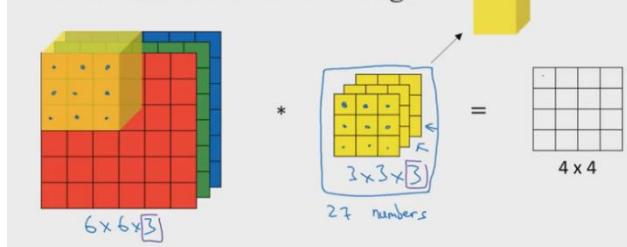
5 convolutions-over-volume 3D filter

5.1 3D filter

image 本身就是 3 维立体
所以 filter 也是三维的
filter 跟 tensor 的厚度一样！！

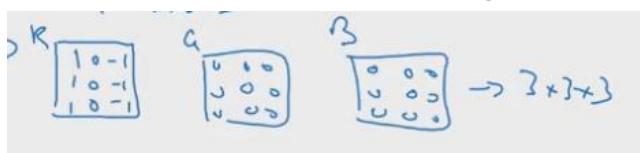


Convolutions on RGB image



只是更多的像素做点乘(立方体的点乘), 对应位置元素相乘求和, output 还是一个值！！最终是个 2 维 tensor

如果只想 detect red channel edges: 只操作一个 channel



其他层的 filter 参数置 0

或者 detect 所有层的 edges

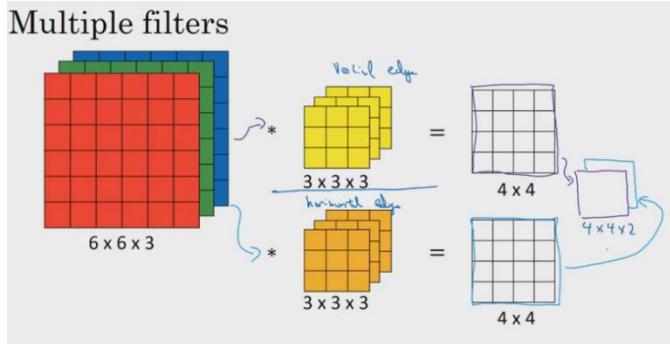


5.2 Learnable filter 本质

filter 本质就是参数，代替之前的 fully-connect 参数！！！

我们想 detect 多种 edge，因此需要多个不同的类型的 filter 来捕捉不同的 feature，每个 filter 会单独产生一个 2 维 output，叠加 stack 在一起得到最终的 3D output！！！depth 从 input tensor 提取特征的个数！！每一个 filter 都需要被学习，因此是参数！！

一个 filter 对应一个特征，一层 output 是特征！！



如果 no padding:

$$\text{Summary: } \begin{matrix} n \times n \times n_c \\ 6 \times 6 \times 3 \end{matrix} * \begin{matrix} f \times f \times n_c \\ 3 \times 3 \times 3 \end{matrix} \rightarrow \begin{matrix} n-f+1 \times n-f+1 \times n'_c \\ 4 \times 4 \times ? \end{matrix}$$

↑ #filters

output 厚度 depth(edge 的种类！！)=#filter 个数！

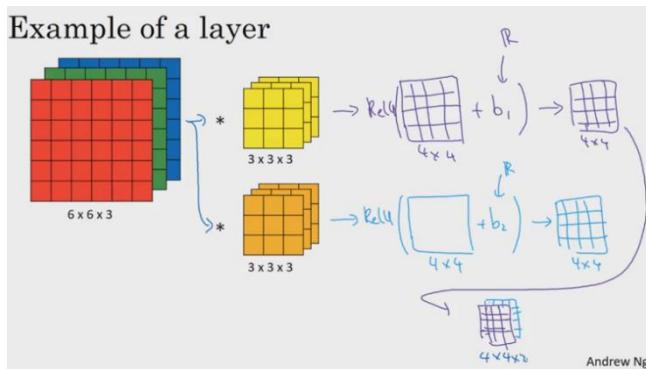
可以看出 filter 作为一套共享参数，使得每一层比 full connect 的参数更少

6 one-layer-of-a-convolutional-network(维度约定！)

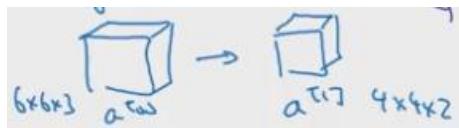
6.1 bias+non-linear activation

1 需要给 output 加个 **bias**(broadcasting，每个元素都加相同的 b 值)elementwise

2 再加一个 non-linear activation(**relu**) elementwise
output 的 size 不变



以上只是一层 convolution layer，一次输出 output：



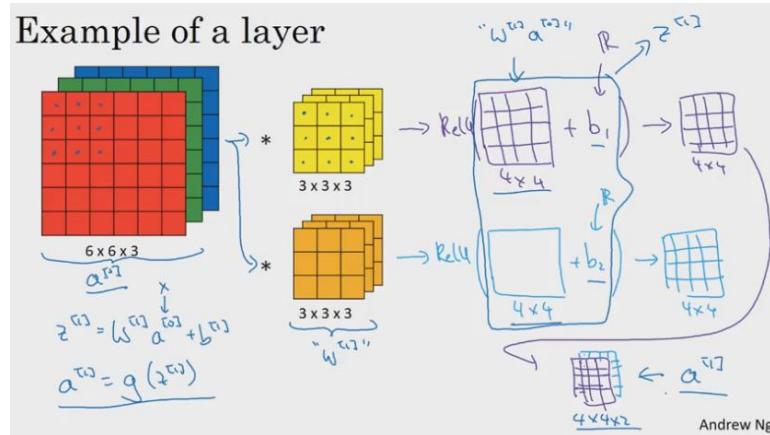
6.2 反向传播？

左下角是我们传统的从 $a[0]$ 到 $a[1]$ 的过程

现在 image 是 $a[0]$, filter 是 w , 卷积过程是 $w * a$ (linear operation), 激活以后得到的 output 是 $a[1]$

此时 filter 就是参数，是学习得到的！！！

Example of a layer

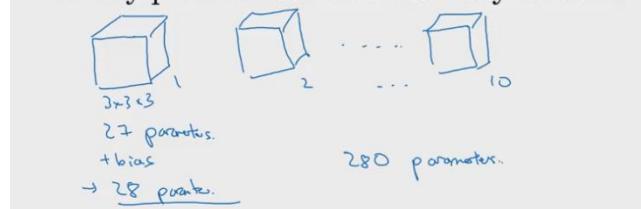


question? 具体推导?

6.3 模型参数个数

Number of parameters in one layer

If you have 10 filters that are $3 \times 3 \times 3$ in one layer of a neural network, how many parameters does that layer have?



参数就是 filter 和 bias

先计算一个 filter 的参数，乘以 filter 个数

维度总结：当前层的维度定义

假设 layer 1 是 convol layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

1 input 和 output (activation) 维度确定

Input: $\underbrace{n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}}_{\text{维数}} \leftarrow$
Output: $\underbrace{n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}}_{\text{维数}}$

$$n_{HW}^{[l]} = \left\lfloor \frac{n_{HW}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l]}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

对于多个 example: 一张照片计算出一个 tensor, m 就 m 维前面

$$A^{[l]} \rightarrow m \times \underbrace{n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}}_{\text{维数}}$$

$$A^{[l]} \rightarrow \underbrace{m}_{\text{维数}} \times \underbrace{n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}}_{\text{维数}}$$

in layer l. $n_c \times n_H \times n_W$

有些会把 channel 放前面

2 weight(filter):

同一层的 input tensor 的 **多个 filter** 的维度应该一样, 因为只有这样, output 维度才一样, 才可以 stack 一起!!!

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
bias: $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$ # if (tos in layer l)

因此参数 weight 最后一维是 filter 个数, **每个 filter 一个 bias** !!!

3 fully connected layer:

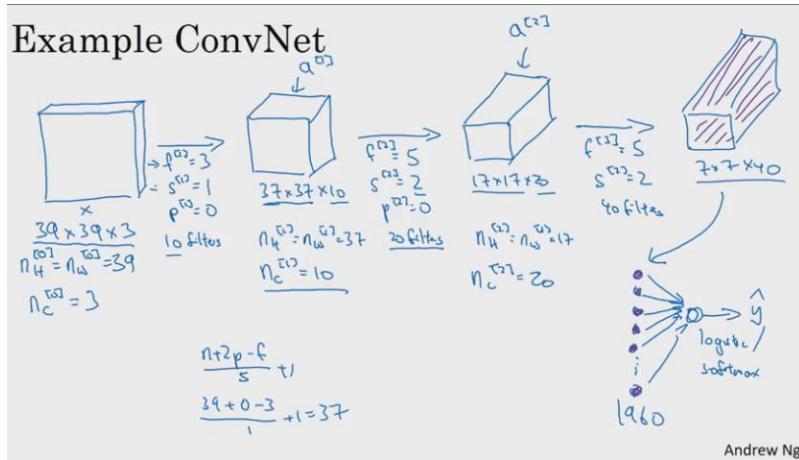
需要将 activation tensor 展开

X = Flatten()(X) # (m, 18, 18, 64) 变成 (m, 18*18*64)

维度有固定的顺序

如果前一层 tensor 不是正方形, w 和 h 要分开计算

7 deep convolutional-network-example



将最后一个卷积层 output, unroll 展开成 vector output
最后连接 final output 节点, 根据实际问题选择节点个数
logistic or softmax

三种 layer:

Types of layer in a convolutional network:

- Convolution (CONV)
- Pooling (POOL)
- Fully connected (FC)

8 pooling-layers

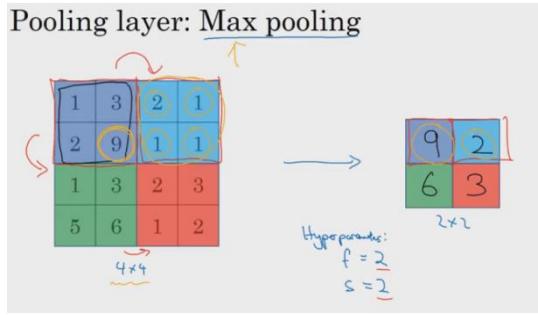
1 Reduce the size of representation to speed up computation (reduce computation)

2 helps make feature detectors more **invariant** to its position in the input. Make feature detect more robust question?

POOL 操作类似于 filter, 也有 size 和 stride (hyperparameter, fixed number), 和 filter 不同:

- 1 没有 parameter to learn
- 2 只有 2 维! 一层一层单独操作
- 3 极少用 padding

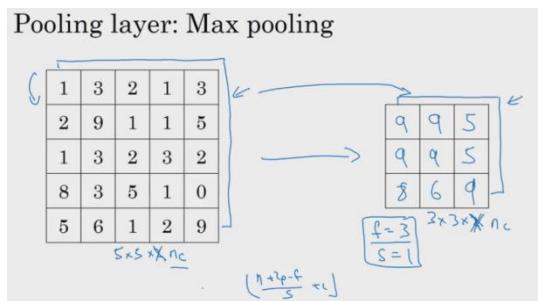
直观理解:



每一次卷积层之后，都要 pooling 下

大的值(更亮白的 edge)代表我们经过卷积操作后，detect 到的 feature，在当前的 pooling filter 里，通过 max pol 保留这个 feature 值(Make feature detect more robust)

如果当前的 pooling filter 里四个值都很小，说明没有我们的 detect feature，就算是保留最大的，也没有我们的 feature。max pooling 其实就是为了保留我们 detect 到的 feature！



相当于将 feature 进一步提纯

max pooling 比 average pooling 应用更广泛

average pooling 用在 very deep network 层

$7*7*1000 \rightarrow 1*1*1000$

Summary of pooling

Hyperparameters:

f : filter size $f=2, s=2$
 s : stride $f=3, s=2$
Max or average pooling

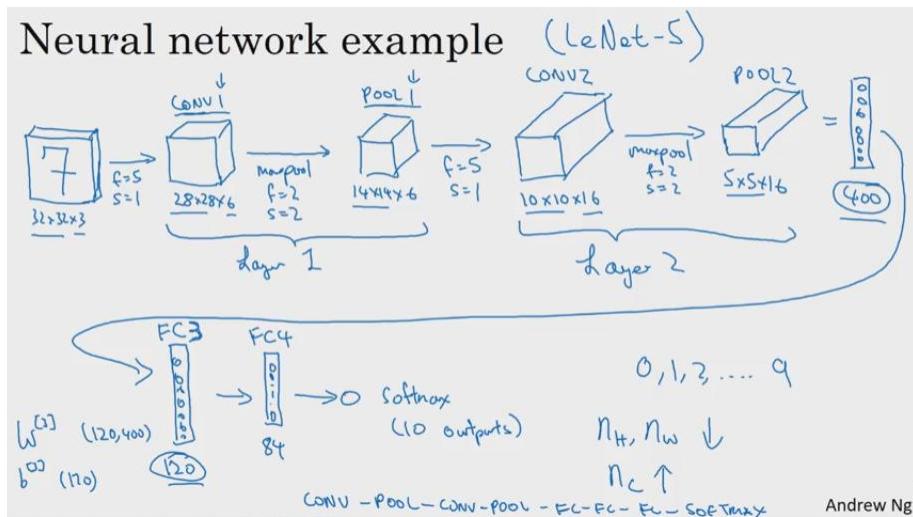
$$\begin{aligned} & N_H \times N_W \times N_C \\ & \downarrow \\ & \left\lfloor \frac{N_H - f + 1}{s} \right\rfloor \times \left\lfloor \frac{N_W - f}{s} + 1 \right\rfloor \times N_C \end{aligned}$$

计算 pooling 结果

9 cnn-example LeNet-5(卷积网络的作用本质)

不要自己设定 hyper parameter，要参考文献的经典参数

Neural network example



pool layer 是对前面 convol tensor 一层一层的操作，depth 不变
上面每个立方体都是一个输出，有 conv 输出和 pooling 输出

- 1 通常会把 convol + pooling 称作一个 layer(有参数的才作为 layer，所以一般不会把 pooling 称作一个 pooling layer，但实现上是个 layer)
- 2 unroll 最后一个 convol 层输出，连接 fully connect layer
该层参数个数是个矩阵[后一层 unit 数量，前一层 unit 数量]
- 3 每一层的 output 输出，depth 增加，w, d 减少！

本质：

捕捉到的 feature 数量越来越多 (depth 增加)
并且 feature 不断的进行提纯 (w, d 减小)

Summary:

Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Andrew Ng

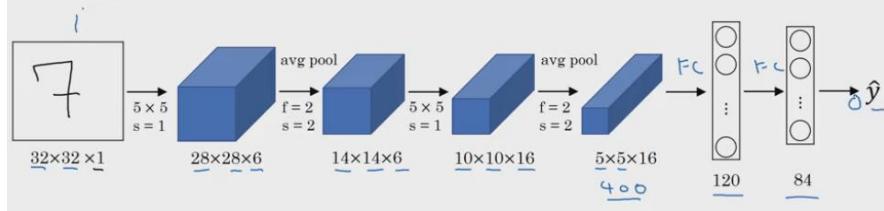
可以看出每个立方体的维度和包含的像素个数，参数主要集中在 **fully-connected layer** (前一层节点数 * 后一层节点数)

activate size 但如果 drop 太快，也不利于模型的表现

关键在于如何利用以上的基本组件，来组建一个有效的神经网络。

多看成功的例子！！

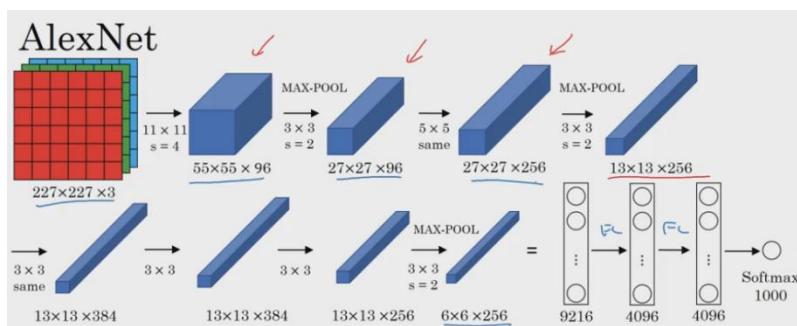
LeNet - 5



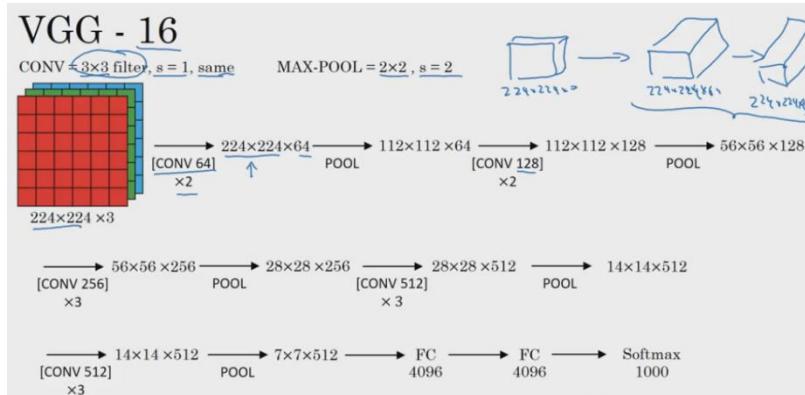
最后连一个节点，输出 0–9, sigmoid/tanh

现在，我们最好用 10 个节点的 softmax 和 relu

AlexNet:



use relu



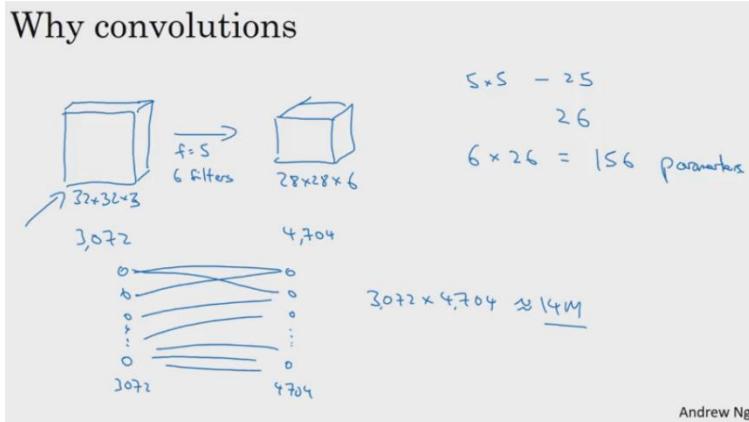
simplify architecture

用固定的超参数

1 先两层 conv 2 再 pooling

filter 数量每一 step double 一次

10 why-convolutions



如果还是按传统的 network，参数太多了！！！

convolutions 2 个 advantage:

Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

Sparsity of connections: In each layer, each output value depends only on a small number of inputs.

Andrew Ng

1 parameter sharing

有时我们想在图上任意位置寻找相同的特征

一个 filter 可以理解为一个特定 feature detector

2 sparsity of connection

输出层的每一个 node 连接 input 的一片像素点，而不需要连接所有 (fully-connect) input 像素，稀疏连接

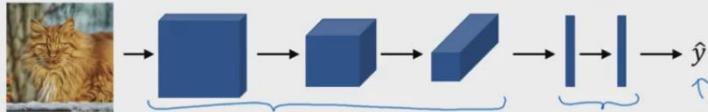
3 少了很多参数，可以适用于小的 train set，不容易过拟合！！

4 translation of variance: 特征偏移或旋转，也认为是同一个特征，原始图片一个猫头旋转，也会被 assign same label！！！

Putting it together

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.

$\underline{\omega}, b$



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce J

11 Resnets

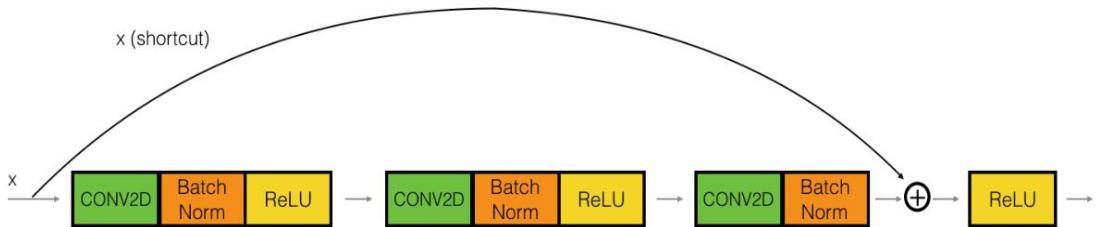
11.1 Residual block: Skip the layer(一定见作业！)

1 A huge barrier to training very deep networks is vanishing gradients: often have a gradient signal that goes to zero quickly, thus making gradient descent unbearably slow

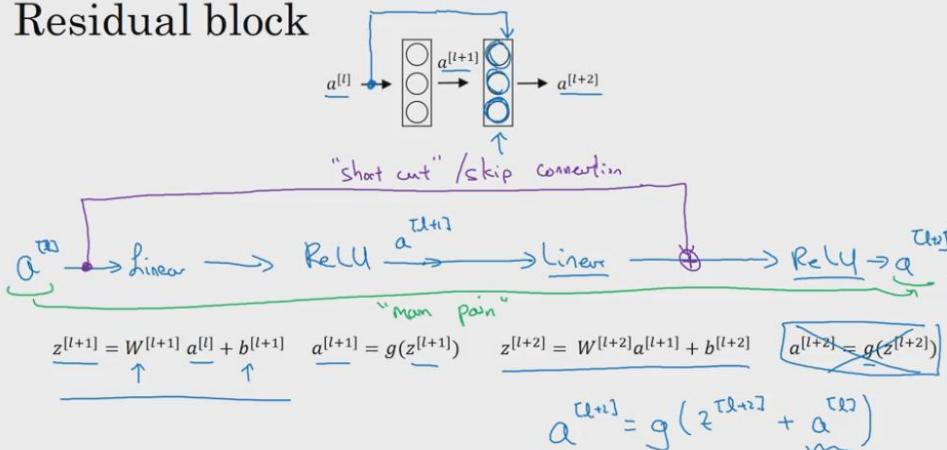
In ResNets, a "shortcut" or a "skip connection" allows the gradient to be **directly backpropagated** to earlier layers:

2 ResNet blocks with the shortcut also **makes it very easy for one of the blocks to learn an identity function**. This means that you can stack on additional ResNet blocks with little risk of harming training set performance, which accounts for **ResNets' remarkable performance**.

skip 3 layers



Residual block



[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

对于两层的普通神经网络：

注意看右下角公式的变化： $a[1]$ 的值加到了 $z[1+2]$ 上(element wise add)
 $a[1]$ 是个 residual 在激活之前 add。

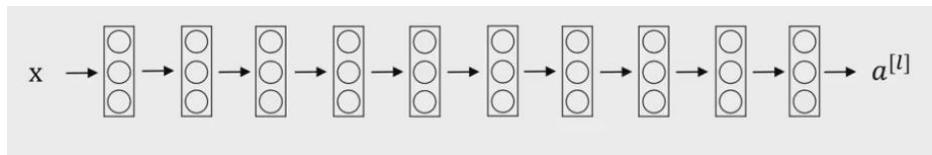
$z[1+2]$ 和 $a[1]$ 维度要一样！！！

如果不一样：

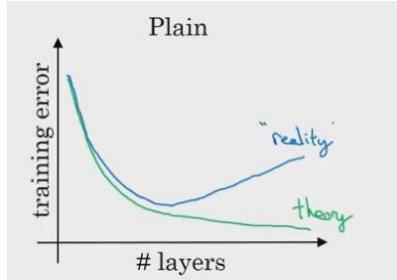
a[1]需要先乘以一个矩阵，这个矩阵值可以 learn，也可以固定值，也可以用0来填充满a[1]到a[1+1]的维度

以上是一个 Residual block 可以帮助我们训练更深的网络！！

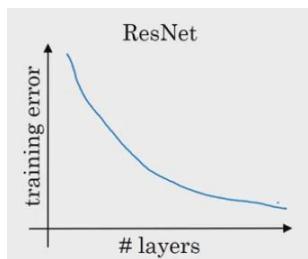
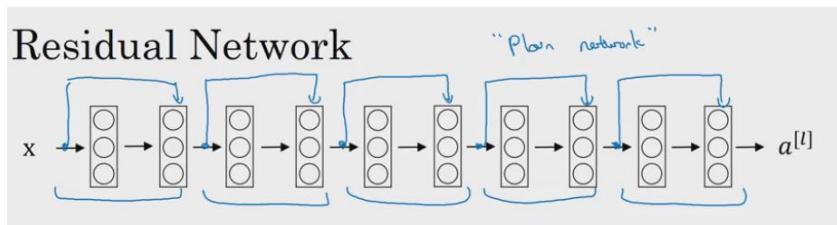
plain network:



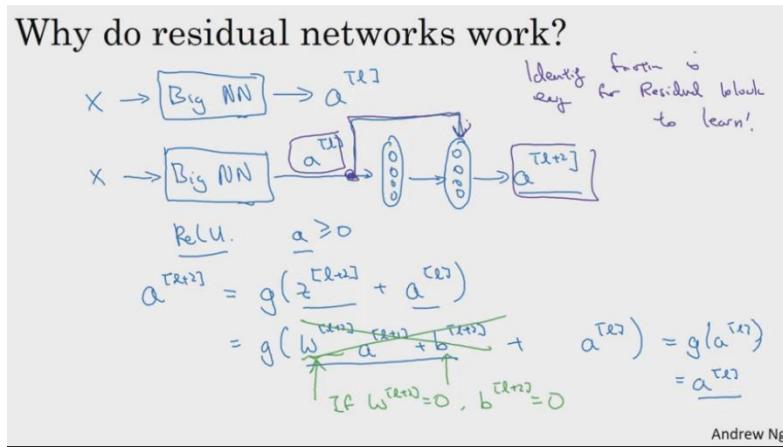
理论上层数越多，准确率效果越好，但实际相反



五个 residual block: 可以训练更深的网络，并且准确率会一直提升！！



11.2 why-resnets-work



我们将 $a[1+2]$ 的 $z[1+1]$ 展开

因为用了 L2 regularization, 会使得 w , b 值很小, 假设接近 0

我们用了 relu: $a[1+2]=a[1]$, residual block 成为一个 identity function
很容易学习到 identity function

结论：

即使增加了两层网络，也会像没有加之前一样(相同的 a 值)，是一个 identity function 只是 copy 一下 a，**不会 hurt the performance**

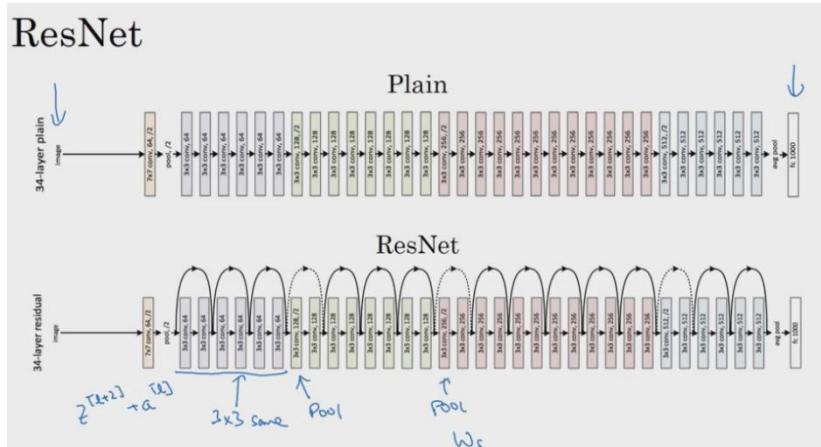
如果参数不是 0 的情况下，会 performance 比 identity function 更好，identity function 就像一个 baseline，只能比它更好。

So Easy to learn identity function，增加两层，没什么变化，至少保证不会 hurt performance

这两层 Hidden layer，一旦参数 w 不是 0，**也就是学到了东西，可能更能提高 performance**

因为 residual 和 z 维度要相同，因此需要用 **same padding** 操作，使得 output 尺寸不变

ResNet



都是用 3×3 filter same padding，因此维度不变

使得 $a[1]$ 和 $a[1+2]$ 维度一致可以加

中间会插入一些 pooling, pool 后，需要重新恢复到维度，可以乘以一个 matrix 或者 padding 0，保持维度一致

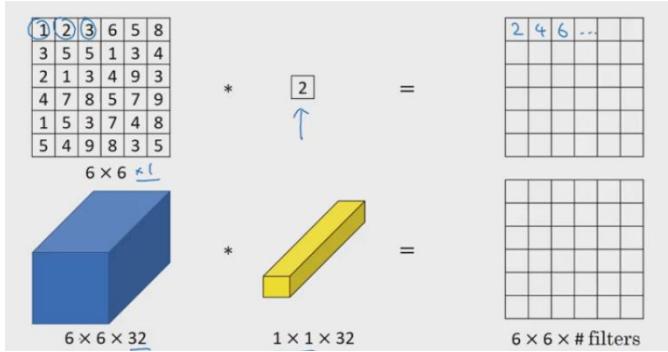
或者用卷积改变维度！！！见作业@！！

question convolutional_block input 和 output channel 不匹配的！！！！

12 inception-network

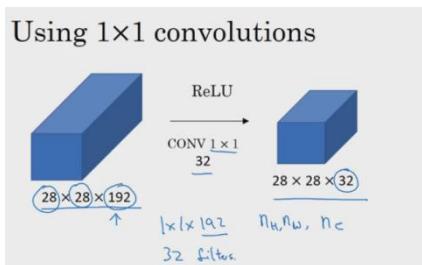
12.1 1x1-convolutions

Network in network(论文)

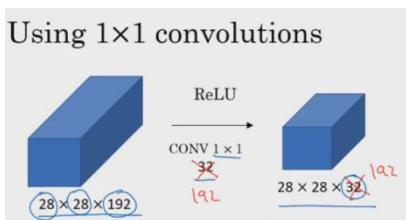


output 尺寸不变，但 filter 的数量决定 depth(channel)

1 可以 Shrik the channel, 不影响 performance



2 也可以 channel 不变，单纯的增加 non-linearity(点乘后，relu)，增加学习到的函数复杂性



12.2 inception-network intuition

Inception networks incorporates a variety of network architectures (similar to dropout, which randomly chooses a network architecture on each step) and thus has a similar regularizing effect as dropout.

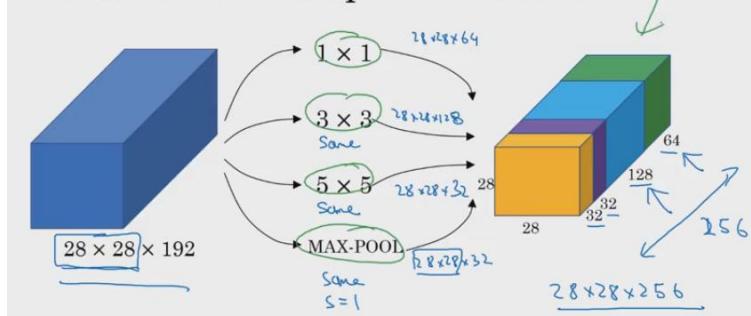
[Szegedy et al. 2014. Going deeper with convolutions]

我们想达到两个目的：

1 make architecture more complicate 2 也同时提高性能

inception module:

Motivation for inception network



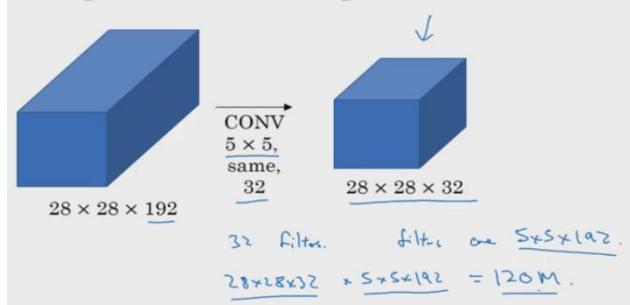
思想：

对于一个 tensor 我们想有多种不同的 layer 去尝试，那就把我们想试的全部 layer 都给 input tensor，让神经网络自己学任何他想学的参数，（不需要我们一个个去尝试了）

让最终得到的 output size 一样，可以 stack 成新的 tensor
conv 都用 same，pool 需要填充 padding，才能保证 size 不变

以上就是个 inception module
但计算量太大了

The problem of computational cost



计算量太大 120 million 乘法

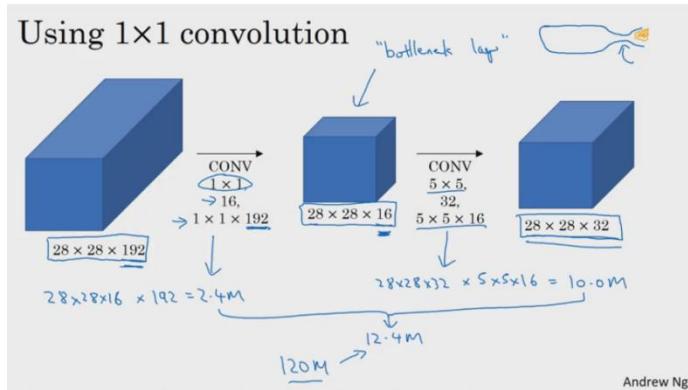
解决：

先用 1*1 shrink，创建 bottleneck layer

不会 hurt performance，节省太多计算！

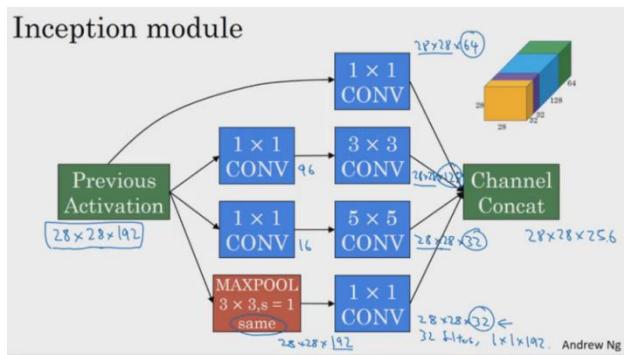
bottleneck layer

类似于瓶子的最细部分(代表 shrink)



12.3 inception-network google net

一个 inception 模块：



在尝试多种 layer 之前先 shrink channel，减少计算量

第一个是直接 1*1 缩到 64 channel

第二，三个是 1*1 缩到 96，然后再 same padding

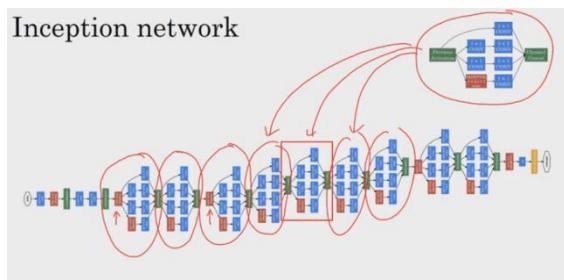
第四是先 same pooling，channel 还是很多，再 1*1 减少 channel

以上全部的 size 一样，只是最终 channel 不同

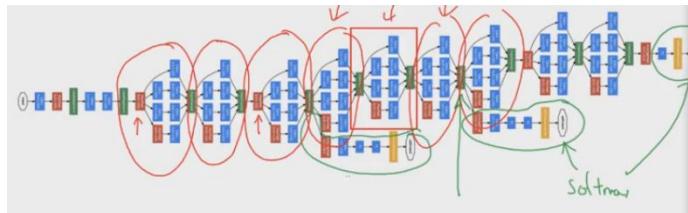
concatenate 所有结果

inception-network

是多个 inception module 组合在一起！！



论文中又增加了新的模块：side branch 分支，是一个 softmax 用来 predict label，检验和保证，在 hidden layer 就可以得到较好的结果！！



起到 regularization, prevent overfitting(避免模型过于复杂), 一点准确率低, 就减少结构, 不再 stack

13 practical-advice-for-using-convnets

13.1 using-open-source-implementation

最好是从别人的 open-source-implementation 开始, 而且可能是 pre-trained 模型, 省了我们再 train 了

Github

复制 github 链接

代码

Git clone 链接

```
C:\Users\Andrew Ng>git clone https://github.com/KaimingHe/deep-residual-networks.git
```

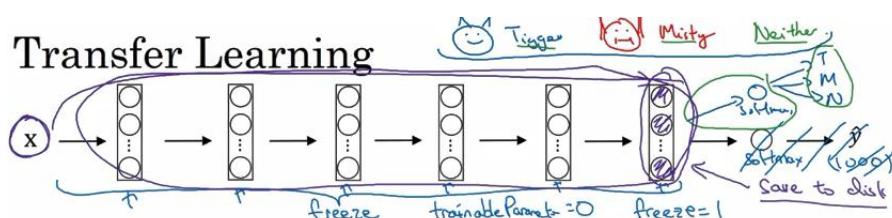
会下载到本地目录下

13.2 transfer-learning

Pretrained network transfer to 不同的 dataset

假如想做特定几只猫的分类, 但我们的 train set 很少

Download the code 和 weight of nets

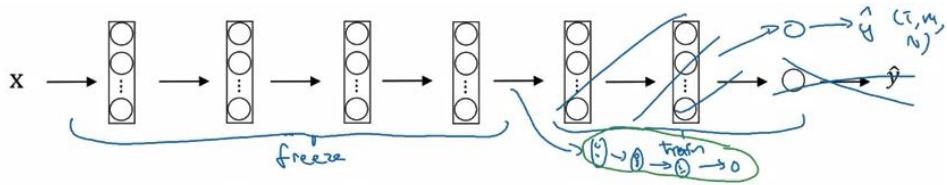


1 将原网络的 softmax 去掉, 换上我们的

2 Freeze 前面所有层的 parameter(fixed function), 只训练与我们 softmax 直接连接层的参数

因为前面是个 fixed function, 也可以先计算全部 input train set 的 forward 最终值, 也就是最后一层 activation, 存在硬盘, 然后作为输入, 来训练 softmax 参数(分两步做了)相当于一个简单的神经网络

如果 train set 多, 不仅可以 train softmax layer, 还可以 train 更多 layer



freeze 前面几个 layer, train 后面几层 layer, 或者直接去掉后面几个 layer, 换成我们的新的 hidden layer! ! !

如果更多 data

将 pretrain 所有参数作为 initialization, 去训练全部网络

尽量用: 现成的, 除非自己发明算法

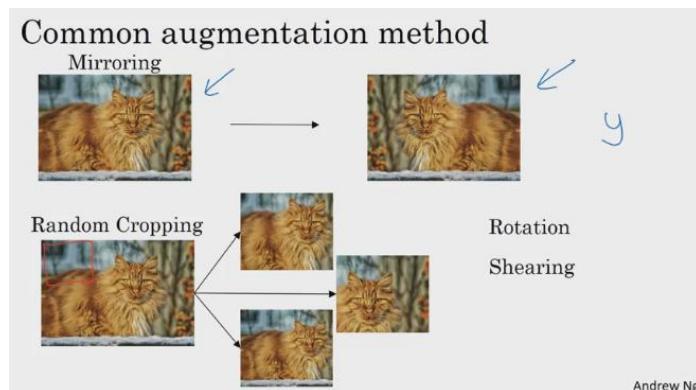
Use open source code

- Use architectures of networks published in the literature
- Use open source implementations if possible
- Use pretrained models and fine-tune on your dataset

13.3 data-augmentation

可以用一些开源 code

两种最常用:

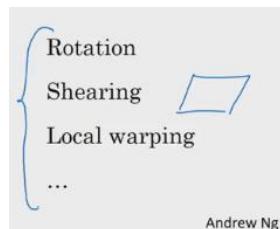


Andrew Ng

random cropping 并不是最好的选择, 可能随机到不需要的部分

只要保证是有效的部分就好

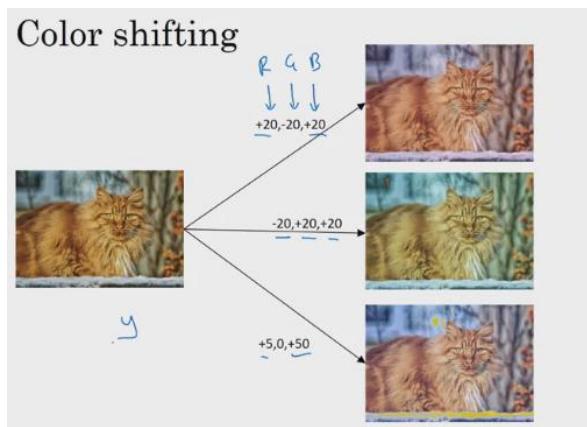
下面的用得少:



Andrew Ng

另一种用的多: Color shifting

draw R G B FROM distribution , distort the image, label 还是一样

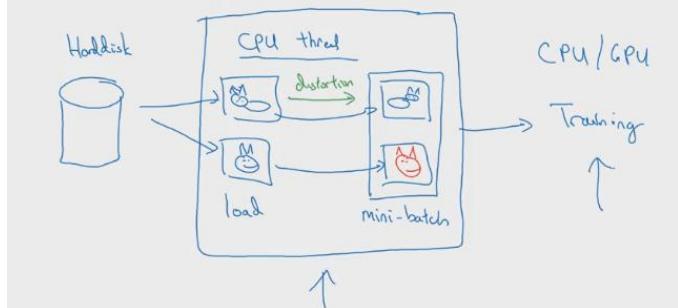


make algorithm more robust to color change

sample color: PCA color augmentation

假如原来 image 包含 R,G 多 B 少, 会增加或减少 RG 更多, B 增加或小, 按原来的比例来, 保证整体的 color 一致, 不会变的太夸张

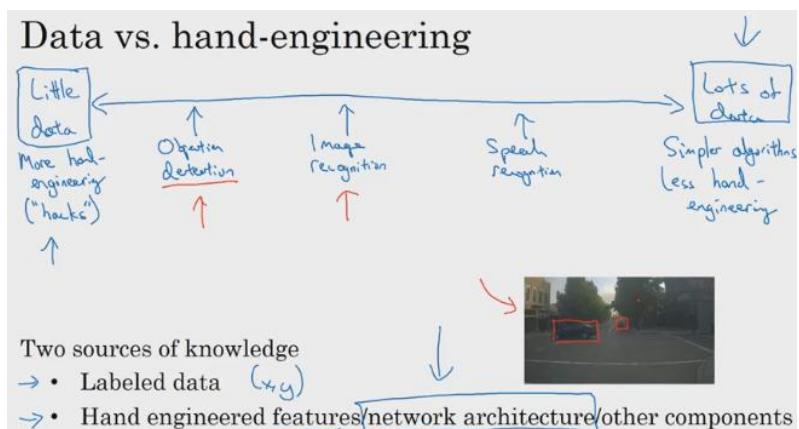
Implementing distortions during training



多线程读取数据完成 distort 图片, 形成 mini batch, 然后给训练模型, 读取和训练是并行的!!

13.4 state-of-computer-vision

目前, 不同问题拥有的不同数据量



数据量大的话, 可以用相对简单的模型, 需要较少的 hand-engineering

hand engineering, 手动提取特征，手动调结构，组件

computer-vision 目的是学习非常复杂的函数模型，因此数据往往不够，更多依赖 hand-engineering，用非常复杂的结构模型，**因为缺少 data，只能从结构上下手**，也可以用 transfer learning

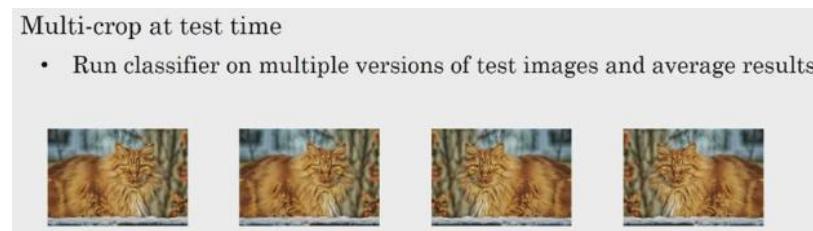
13.5 do well on benchmark

算法在 benchmark data 上如何**拥有更好的表现！**

1 ensemble: 将多个算法结果平均: mean of \hat{y} !!!

会比 benchmark 高 2% 准确率，但运行效率低计算量大，实际应用少

2 apply data augmentation to test set，让分类器在不同的类型的 test 跑，再**平均下结果，可以提高准确率**



先 copy 4 张图片，12 一组 34 一组

对于 12

1 每次在第一张照片选取一个中间区域 crop，作为第一类 test

2 再在另一张同样图片选取四个角 crop size 不变，4 个 test

然后再 3,4

得到最终 average test 结果



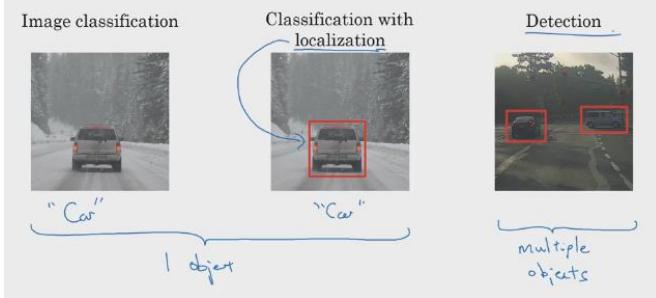
crops image

14 object-detection

14.1 object-localization(detection)

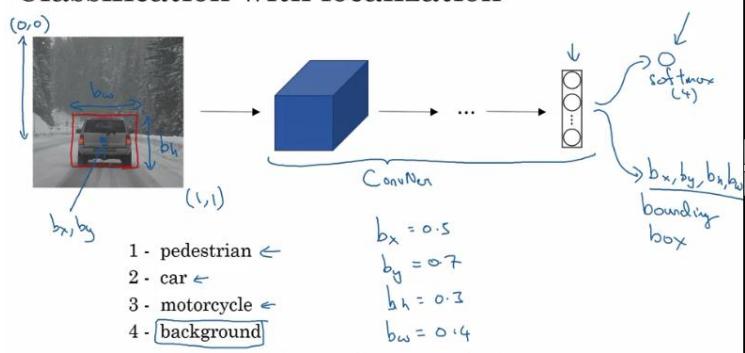
只需要在原先基础上，增加一些 output unit(**位置信息**)

What are localization and detection?



分类是基础，然后分完类再定位

Classification with localization



在 conv 的 softmax 节点基础上(分类基础上)，增加 box 的四个坐标节点可以自动学习到 object 的类别和边界！

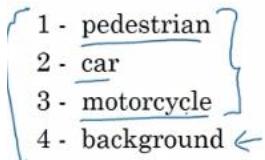
假设图片左上角坐标 00，右下角左边 注意这个坐标是倒的 横 x 坚 y
边界盒子 bounding box 参数：

bx by 是盒子中心点坐标

bh bw 是盒子的高和宽

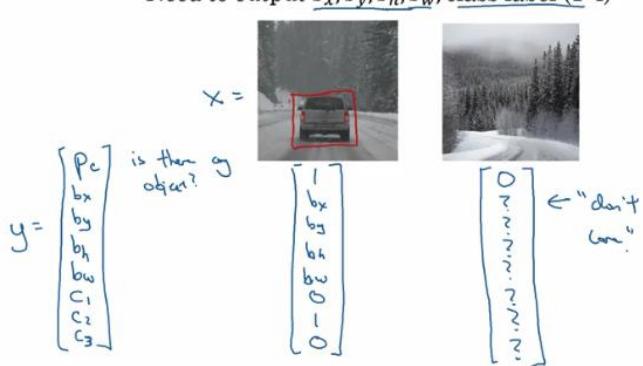
模型不仅输出 y label，而且输出四个参数

假设：一张照片一个 object 一共 3 类 object，剩下是背景



定义 y 向量：

Pc: probability of one class 是否是 object 的概率



先分类，是否是我们四个类别的一类？

如果 $pc=1$, 需要指定哪一类(只有我们关心的三类 object)

如果 $pc=0$ (没有 object), 不用管, 用? 代替, 不需要 box 和类别

所以计算 loss 时, 要分两种情况: 当真实 label y $pc=1$ 和 $pc=0$, 定义两种 loss, 再相加! ! !

是要定义两个 cost 吗? ? 还是直接相加 ? ? question

loss func:

$$l(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ + \dots + (\hat{y}_n - y_n)^2 & \text{if } \underline{y_i=1} \\ (\hat{y}_1 - y_1)^2 & \text{if } \underline{y_i=0} \end{cases}$$

$y=1$, 全部元素的平方和

$y=0$, 只做 pc 的平方和, 只关注 pc , 让 pc 正确, 不需要分类定位
或者也可以:

c1, c2, c3 用来做 softmax ;

bx, by, bh, bw 做 squared error

pc 用 logistics regression

$$y = \begin{bmatrix} \rightarrow & pc \\ \uparrow & \left[\begin{array}{c} bx \\ by \\ bh \\ bw \\ c1 \\ c2 \\ c3 \end{array} \right] \end{bmatrix}$$

可以让 CNN 自动去学习!!!

mix cost func 具体如何实现? question

14.2 landmark(关键位置)-detection

只需要增加一些 output unit, 来输出 landmark 坐标

脸上的坐标位置: landmark (关键位置)

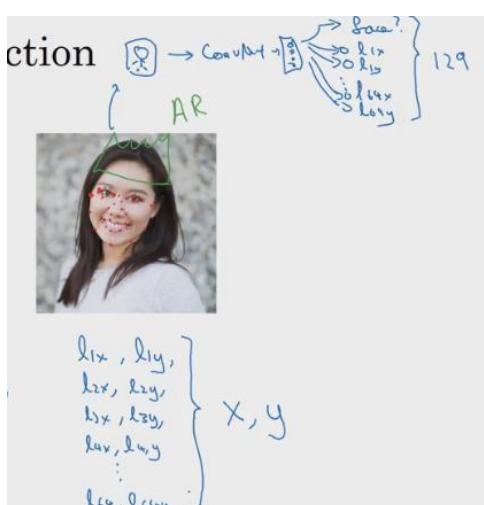
1 corner of eye (11x , 11y) (12x 12y)

两个眼角, 只需在最后一层多加 4 个数值节点

2 也可以多个点: along the eye, 可以通过 shape 来预测人是否笑

number of landmark 产生 label training set

y 的话需要, 人工的标注和记录(比较繁琐)



label y:

1 第一个节点, 是否为 face

2 是的话, 后面全是 key landmarks(关键位置)

输出的 lanmark, 可以应用捕捉 emotion, 以及脸部特效

自动捕获脸部关键部位

人的肢体上寻找 key landmarks

来分类不同 position



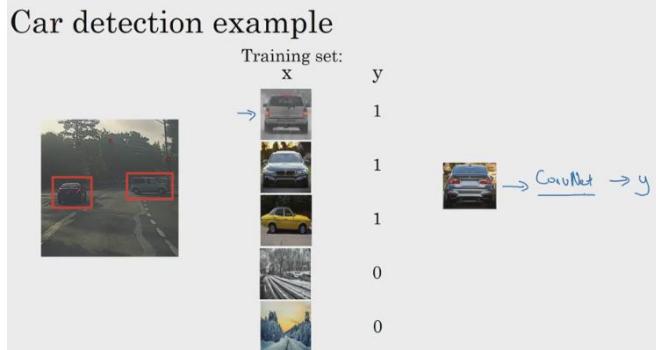
所有图片的 lanmark 位置, 都要 consistent

14.3 object-detection

sliding window

1 先 **closely cropped image** 作为训练数据(单纯的 **centered object** 图片)得到卷积神经网络 conv, 得到训练好的神经网络!!! 可以识别所有分类!!

Car detection example



2 sliding window detection:

2.1 pick a window size, 在原图上滑动, 然后给 convnet, 得到 label y
就像 filter 一样滑动

2.2 再用更大的 **window size**, 滑动

目的是: 总会有一个 window 可以框住车, 并输出 y

然后看最终的输出 y 的结果, 就知道图片有哪些 object!

disadvantage: 计算量太大!!

但对于每 slide 一次都要进行卷积神经网络计算, 计算量太大了!!

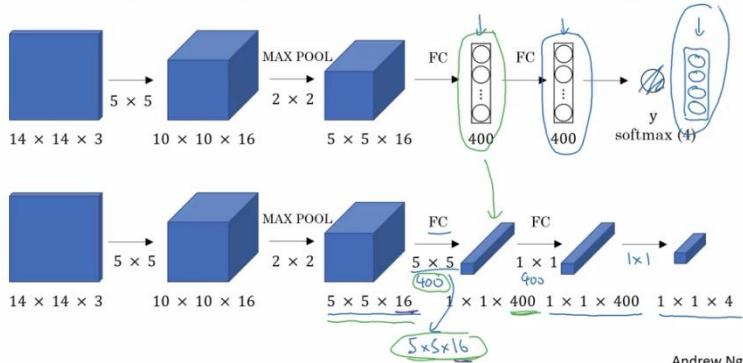
用 convolutional-implement-sliding-windows

通过卷积操作, 一次性得到所有 slide window 的结果

可以让一张照片的所有 slide window predict at same time, 得到全部结果

Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Turning FC layer into convolutional layers



1 先改装我们的之前的卷积网络结果, 再重新训练!

卷积和 fc 的转换: (注意不是 1*1 conv)

用卷积来代替 fc: 用和 input(5*5*16) 维度一样的 filter (5*5*16) 做卷积
得到 size w, d=1 的 output, 进行 stack! !

数学上等价：

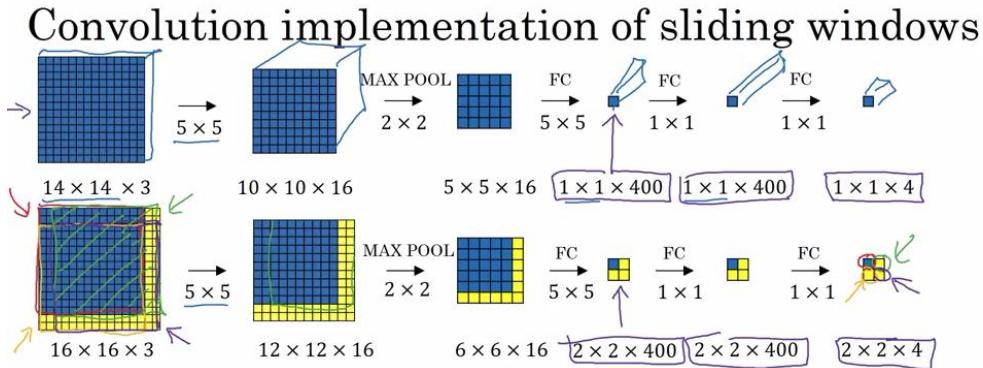
只考虑一个输出节点情况：

之前 full connect，每个节点参数是 $5 \times 5 \times 16$ 个

现在卷积 filter 就是参数！一个 filter 也是 $5 \times 5 \times 16$ 个参数。

且连接的 input 相同，因此是等价的！！！

2 卷积操作实现 slide window：



1 第一行，假设我们的 slid window 是 14×14 ，最终得到 $1 \times 1 \times 4$ 结果
用 closely cropped image 14×14 来训练这个 convnet

2 test set 是 16×16

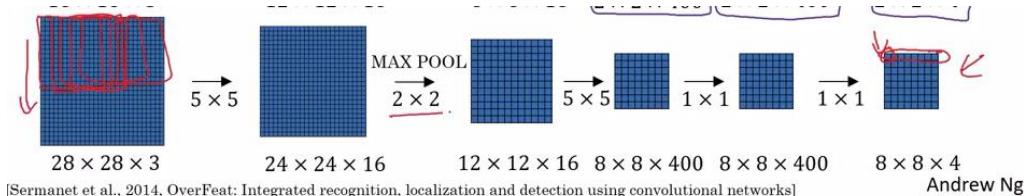
传统做法是，用 14×14 sliding window stride=2，给上面的 conv 得到 label，
但计算量太大，而且很多重复的计算部分(4 个 window 重复)

现在做法：用上面 convnet 同样的 filter(5*5)，max pool(2*2) 对 test input ($16 \times 16 \times 3$) 进行操作，只是后面得到的 output 维度不同

最终得到的四个方块的颜色就刚好是 这四个 slide window 的 label

最终每个块的值就是一个对应 slid window 的输出结果

例子：



最后 output 每个格子代表一个 slide window 结果

也就是 convnet 的最初的 closely cropped image 就是 slide window 的 size，
而输出结果的维度决定 slid window size 和 filter stride 的大小

test 图片 slide window 方式由什么决定？卷积的扫描的方式 (filter 大小 stride)

因为上面如果不是 5×5 filter 的话，得到的不是 $8 \times 8 \times 400$
就是另外一种 slide window 方式了！

14.4 bounding-box-predictions YOLO(推出 slide window, stride)

question 唯一的困惑就是 non-max 的具体操作，每次选最大的 pc，下次循环怎么操作？不再选这个了吗？

Output accurate bounding boxes



假如没有一个 slide window 可以很好的框住车，怎么办？

如何得到更精确的边界？

Yolo paper is hard 对于吴恩达都很难，尤其是 detail

Yolo:

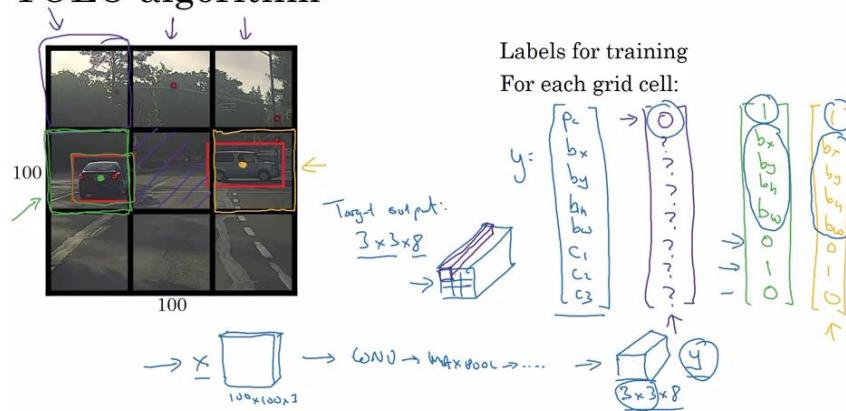
- Input image (608, 608, 3)
- The input image goes through a CNN, resulting in a (19, 19, 5, 85)
- 也就是一张照片 19*19 个 slide window 5 个分类结果，共 85 张照片
- 注意 yolo, slide window 是将图像分割(window 不会有像素重复)
- dimensional output. yolo output

由结果 19*19 推出 convnet 的 slide window 大小和 filter size 和 stride
slide window 就是 CNN 的 input size

然后为了得到 19*19，还要在(608, 608) image 先进行推算，看 CNN 的 slide window, filter, stride 的大小，才能得到我们最终的 yolo output

question 维度之间的关系！！！

YOLO algorithm

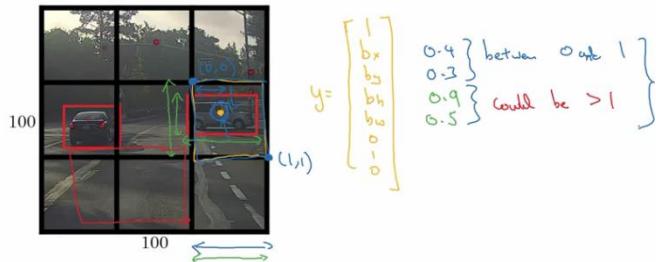


- 1 先确定图片分割大小，每个格子 size
- 2 再按这个格子大小 closely cropped image 进行训练 conv
- 3 分割 input 图片，将每个 grid 放入我们前面训练好的 conv，得到结果
- 4 将每个包含 object 中心点 center point 的格子，label 成 true，代表格子是否包含 object

例如虽然最中间格子包含两个车，但中心点在另外两个格子上
 3 然后对每个格子进行分类和定位算法(前面训练好的 convnet 算法)
 每个格子给 convnet，输出 label+box(边框)

先要对每个格子进行 label(1 是否是分类 object 2 哪一类 3 box 边界)
 如何 label : label the box

Specify the bounding boxes



首先：每个格子 grid(不是整个图片)左上角 (0, 0) 右下角坐标 (1, 1)
 object 中心要相对于上面坐标系 一定小于 1
 object 长和宽要跟 grid 的长宽比较 可能大于 1
 还要 label pc 和 class

因此模型输出 y 是一个向量 (pc, bx, by, bw, bh, c1, c2, c3.)
 pc 是 probability of one class，是概率值！！
 9 个格子，每个格子输出一个 y predict 向量
 共 3*3*8 output

14.5 intersection-over-union IOU

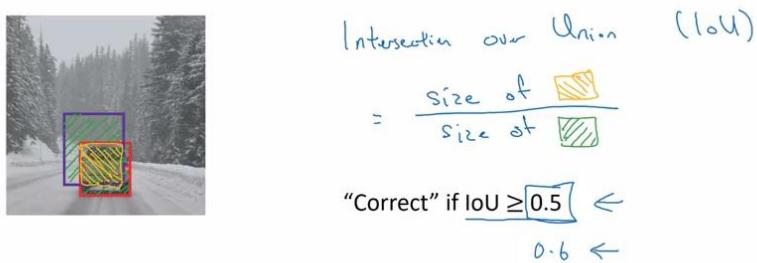
如何评估 localization

Intersection over union

红色框是我们的 ground true bounding box(最理想的 box)

蓝色是模型输出的 box

Evaluating object localization



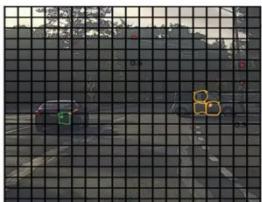
More generally, IoU is a measure of the overlap between two bounding boxes.

Andrew Ng

分子是 intersection, 黄色交叉区域
分母是两个 box 的 union, 0.5 是个 threshold

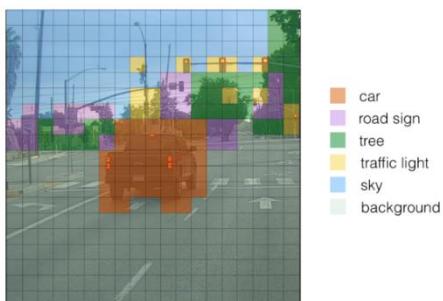
14.6 Non max Suppression

为了让算法 detect one object **only once**(也就是尽最大可能找出 center 落在的那个格子, pc 才为 1)只留一个 box 框住 object! !



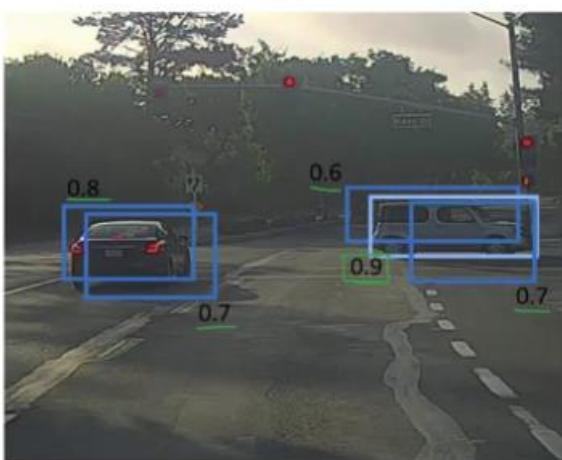
19x19

分割画面，虽然 object center 只落在一个格子，
但每个格子都要 running classification and localization algorithm
每个格子都有输出(pc, bx, by, bw, bh, c1, c2, c3.) **带一个框**
center 附近的格子都可能都被判定为 car, 但我们只需要最佳的一个 box



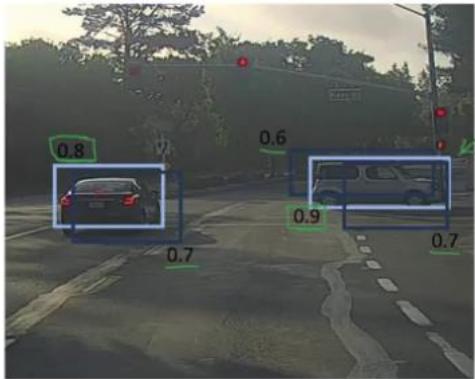
每个格子都独立的 predict, 越临近的结果越像! ! !
出现 **multiple detection of each object**(多个格子判断同一 object)

non-max-suppression 去除其他的留下一个



- 1 先给 pc 一个 threshold , 剔除一些 box
- 2 然后选 pc 最大的 box, 作为 prediction(右边的 car box pc 最大)

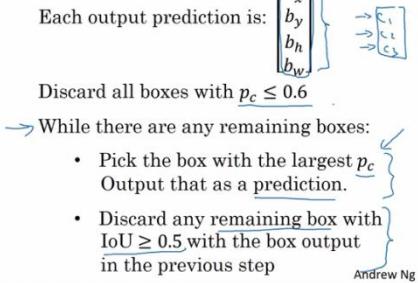
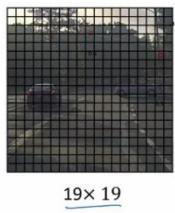
然后遍历所有剩余的 box(包括左边 car 的 box), suppress 那些跟 prediction box 有 high(>0.5) IOU 的 box(也就是剔除一些同一个 object 次等的 box)



基本上右边 car pc 大的都会被去除, 然后遍历剩余的 box(去除前一次最大的 pc box)遍历 pc 最大的应该是左边的 car 的蓝色边框, 再剔除重叠的再循环, 找最大的, 发现没有需要剔除的了, 就结束了!!!

例子:

Non-max suppression algorithm



重要补充: 每次循环 remain, 代表排除了 largest pc 的 box, 从剩余的里面再选最大的 box

经过上面算法应该每个 object 只留下一个最好的 box

上面只是 car, 如果图片有多个 object 的话, 每个 object 都独立的执行 while 循环

14.7 anchor-boxes (one grid include several object center points)

解决: 一个 grid cell predict 多个 object

人和车的 center point 都在一个 grid, 之前还是输出 8 个元素, 就没法区分是哪个 object。

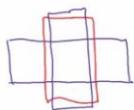
我们用的格子很小时, 发生的概率比较小其实

Anchor box algorithm

Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

$$\text{Output } y: \quad \underline{\underline{3 \times 2 \times 8}}$$



With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

$$\text{Output } y: \quad \underline{\underline{3 \times 3 \times 16}}$$

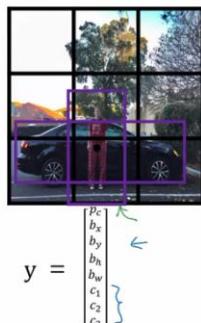
$$3 \times 3 \times \underline{\underline{2 \times 8}}$$

Andrew Ng

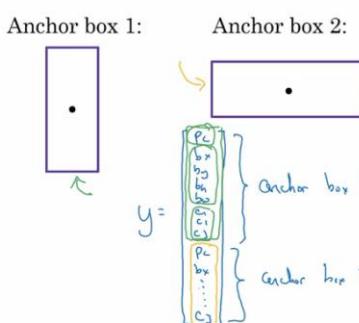
之前是每个 object 对应一个 cell, y 是长度 8 的向量

现在是 1 每个 object 还是会 assign to grid (center point 在的 grid) as before 而且, 2 每个 object 还要 assign to 其中一个 anchor box (IOU 较高那个)

Overlapping objects:



[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]



Andrew Ng

需先预定义 2 个 anchor box, 为了更好的区分不同 object 的形状(每个 object 一个 box) 也就是理想的 box

相应的 y 变成了 2 份长度 16, 此时一个格子可以同时 predict 两个 object!!! 将预测的 anchor box 跟我们预定义的 anchor box 比较(中心点放在一起计算 IOU), 哪个高, 就归为那一类!!

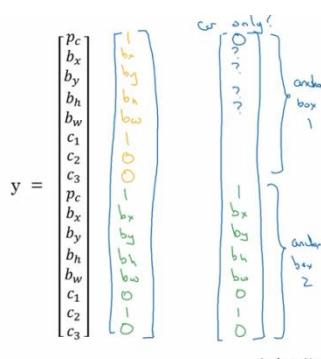
当 center 重合时, 选择与 object IOU 最大的 anchor box

例子:

Anchor box example



Anchor box 1: Anchor box 2:



Andrew Ng

手工 label

辨别用哪个 anchor box 是我们 labeling 时, 靠肉眼辨别的

1 假设最后一行中间格子, 包含两个 object, 是第一列 label(两个 anchor box

都有值)

2 假设某个格子只包含一个 car 的格子，第二列(第一个 anchor box 空，第二个有数值)

如果两个 object shape 一样 这个算法就不行了！！

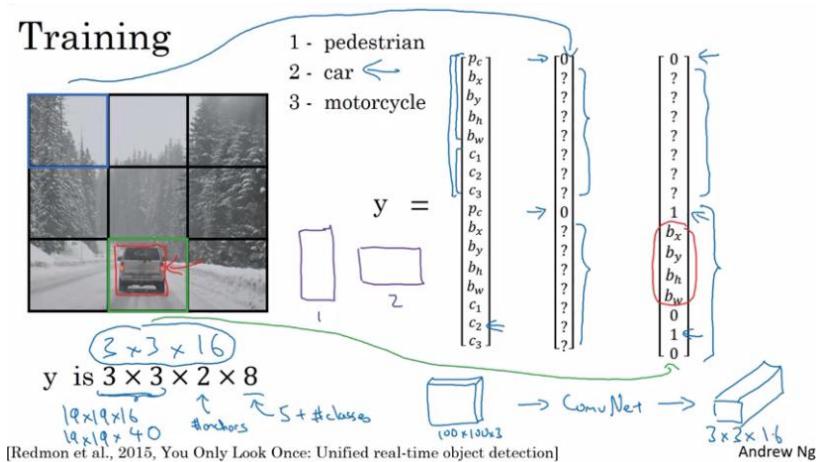
我们用的格子很小时，发生的概率比较小其实

另一种用途：

帮我们 detect 定制化 shape 的 object，高的人，矮的人，胖的，瘦的人

14.8 yolo-algorithm

每个 grid, 因为有两个 anchor, 所以如何取舍？



1 labeling

三个类别，但我们只定义两个 anchor box

制作 train set 时，我们需要遍历每个 grid，都要 label 16 个值
第一个 grid 是中间列，

假设汽车跟第二个 anchor box overlapping 面积最大

绿色 grid 是右边列

最终得到 $3 \times 3 \times 16$ 个 label

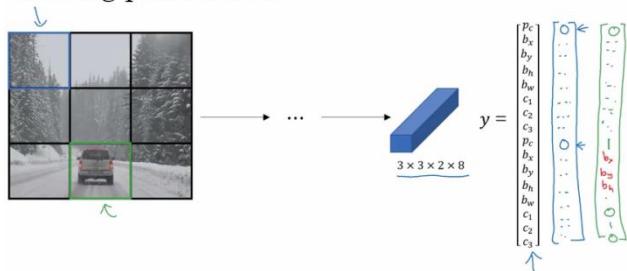
2 training

由 output 可以反推 convnet 的 slide window 的大小和 stride
然后进行 train

3 然后预测：



Making predictions



因为两个 anchor，每个 grid 有两个 box ! !
以上是我们理想状态

3 然后再每个 object 单独的 non-max Suppression
每个 grid 都有两个 box



- For each grid cell, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

需要筛选：

1 先去掉 p_c 小的 box, threshold



2 对于每一类 object 单独用 non-max supression
来预测最终的 box

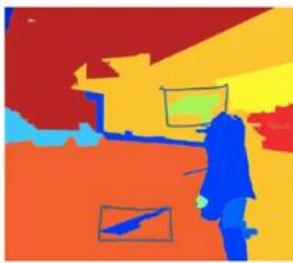


14.9 region-proposals

R-CNN Region with convolution network

只获取一些有效的 region 来 run cnn classification, 而不是 slide window 所有, 因为 slide window 很多重复的都是天空, 会浪费资源!!

先 run segmentation 算法



分割成 block, 然后用上面的两个框框住这些 block 再进行分类, 比直接分成 grid 计算量少很多, 得到 proposed region, 然后用于分类, output label+box 也是一个一个的给 convnet 分类

Faster algorithms

→ R-CNN: Propose regions. Classify proposed regions one at a time. Output label + bounding box. ↩

Fast R-CNN: Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions. ↩

Faster R-CNN: Use convolutional network to propose regions.

Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation]
 Girshik, 2015. Fast R-CNN]
 Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks] Andrew Ng

Fast RCNN 加快分类速度

用 convnet 同时 class 所有 proposed region

先 segment, 再 conv 分类, 但 proposed region 太慢

Faster RCNN(不如 YOLO) 加快 proposed region 的生成:

有时候 Faster CNN 比 yolo 更慢一点!! !

15. face-recognition

15.1 intuition

life detection: 监督学习，分开真实人的脸和照片

Face verification 验证 requires comparing a new picture against one person's face, to know whether the person is the claimed person

给你两张脸问你是不是同一个人

来验证对不对，是不是真人，例机场验证(身份证和脸): 需要出示 id, 通过 id 数据库对应照片，然后和人脸进行比对，一旦没有 id 就没办法验证，而 face recognition 会遍历数据库，找距离最小的！！！

是 1:1 的问题

face recognition 识别 requires comparing a new picture against K person's faces. 给你一张脸和一个库问你这张脸是库里的谁

识别，是从一堆人里识别出来，是 1: k 的问题

⇒ Verification	1:l	99%
• Input <u>image</u> , <u>name/ID</u>		
• Output whether the input image is that of the claimed person		
⇒ Recognition	1:k	
• Has a database of <u>K</u> persons		
• Get an <u>input</u> image	K=100	
• Output ID if the image is any of the K persons (or "not recognized")		

recognition 的基础是 verification

因为 Recognition: 需要对 K 个 person 都 verify, 有 K 次犯错可能

假设 verify 错误率 0.9, recognize 错误率就会很大，要 verify K 次

因此做好识别，必须要先保证 verification 准确率高！！！

下面的内容是为了 built a verification system, 然后可以应用在 recognition 系统里

verification 是 recognition 的组件

15.2 one-shot-learning

训练好的网络，万一突然来个新员工，只有一张照片，是否还要重新训练呢，因为要增加一个 output 节点？

不需要再重新训练，只需 one-shot-learning:

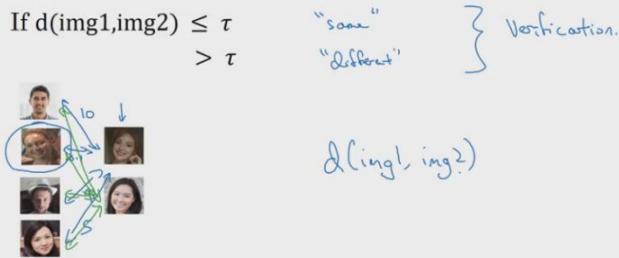
需要学习一个 similarity function, 比较两张照片是否一样

This allows us to learn to recognize a new person given just a single image of that person.

这样人脸识别以后，就可以跟 db 的所有照片进行相似度计算，返回最相似的人！

Learning a “similarity” function

$d(\text{img1}, \text{img2})$ = degree of difference between images



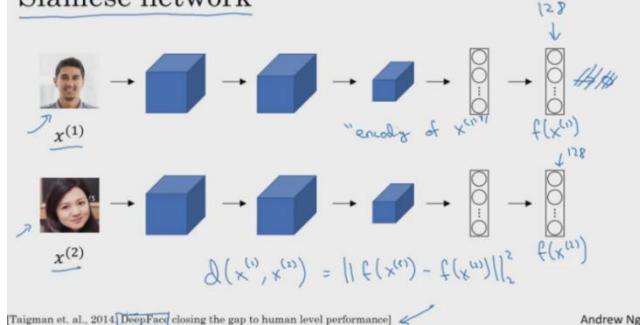
得到两张照片的差异，设置一个 threshold，低于就认为是一个人
以上就解决了 Verification 问题

只要能够比较两张照片的距离就能解决 verify!!!

15.3 Siamese-network 来实现相似函数(embedding)

让 network 去 learn similarity function

Siamese network



用两张照片 feed 到同一 deep net, 去掉 softmax 层, 只留最后一层 FC 的向量值: $f(x)$ (encoding of x)

这个网络叫 Siamese network 其实就是将 image encoding! embedding!!

然后照片的 similarity 是两个 FC 向量的 norm

如何训练这个 Siamese net?

cost function: 是一种 similarity(也就是距离)

须兼顾两种情况: 1 照片同一人距离尽可能要小 2 照片不同人距离尽可能大, 应该包含两个式子!!

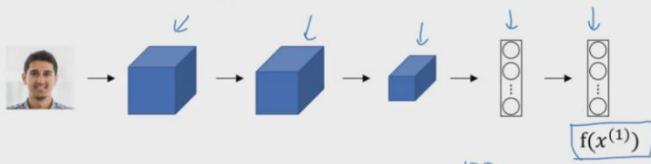
如何将距离转成 cost 函数?

一种思路: 当 positive 时, 计算的距离大于某个 threshold 的值作为 loss, 小于 threshold loss=0 (用 max 函数实现) $\max(d-\text{threshold}, 0)$

当 negative 时, 计算的距离小于某个 threshold 的值作为 loss, 大于 threshold loss=0

这样需要我们指定两个 threshold!! 比较难, 有没有更好的方法?

Goal of learning



Parameters of NN define an encoding $f(x^{(i)})$

Learn parameters so that:

If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large.

Siamese-network 的参数决定了 $f()$ -- 最终 encoding 值

$f(A)$ $f(P)$ $f(N)$ 代表我们最终的 encoding

通过训练来学习到该参数，从而输出的是每个图片的特征向量
再用向量的距离代表图片的相似度

15.4 triplet-loss 定义 cost function

为了 learn encoding 需要定义 cost!!

learning objective:

anchor 和 positive 的相似度大， anchor 和 negative 相似度小

需要三种数据 example

triplet-loss 是同时考虑三种数据：

将我们的 train set 变成一系列的 triplet train set (三种数据类型)

1 每人一张 anchor image

2 每个人的 image(1-多张) positive 一个人至少两张照片

3 其他人的 image 选择性的作为 negative

然后 train 是 A,P,N 全部不同的组合!!!

Anchor	Positive	Negative

1 首先要为这个人选一个 anchor image

Learning Objective

Anchor A Positive P $d(A, P) = 0.5$

Anchor A Negative N $d(A, N) = 0.7$

Want: $\frac{\|f(A) - f(P)\|^2}{d(A, P)} + \alpha \leq \frac{\|f(A) - f(N)\|^2}{d(A, N)}$

$\frac{\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha}{\text{margin}} \leq 0$ $f(\cdot)_y = 0$

Schroff et al., 2015, FaceNet: A unified embedding for face recognition and clustering | Andrew Ng

$f(A)$ $f(P)$ $f(N)$ 代表我们最终的 encoding

我们希望 $\|f(A) - f(P)\|^2$ 距离小 $\|f(A) - f(N)\|^2$ 距离大

但很难给个绝对的距离值，因此用相对方法：

A 和 P 的距离要小于 A 和 N 的距离，

也就是 $d(A, P) < d(A, N)$ (上图不是分子)

也就是：

$$\frac{\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2}{\text{margin}} \leq 0$$

但有可能 $f()$ 一直生成 0 encoding，这种也满足条件，要排除掉，需要要求更严格，加个 margin！！

$$\frac{\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2}{\text{margin}} \quad \text{一定是负的}$$

再增加个整数 alpha 还让它负，也就是需要负的更厉害

$$\frac{\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2}{\text{margin}} + \alpha \leq 0$$

alpha 也是个 hyper-parameter，也是 margin，让两者差距更大，保证准确性也

$$\|f(A) - f(0)\|$$

因此定义 loss function: 同时用到三张照片 A P N (因为要计算两个距离)

Loss function

Given 3 images A, P, N :

$$L(A, P, N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$$

$$J = \sum_{i=1}^m L(A^{(i)}, P^{(i)}, N^{(i)})$$

A, P

Training set: 10k pictures of 1k persons

Schroff et al., 2015, FaceNet: A unified embedding for face recognition and clustering]

增加 max 函数目的，变成惩罚函数：

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha$$

一旦为正就惩罚(越正 loss 越大)，为负就不惩罚

所有的 triplet examples 的 loss 加总得到 cost func J

要完成训练，需要保证一个人至少两张照片，至少可以保证有 A, P(因为 N 我们肯定有)如果只有一个人的一张照片，就不能训练

如何选择 triplet，生成 train

Choosing the triplets A, P, N

During training, if A, P, N are chosen randomly,
 $d(A, P) + \alpha \leq d(A, N)$ is easily satisfied.

$$\|f(A) - f(P)\|^2 + \alpha \leq \|f(A) - f(N)\|^2$$

Choose triplets that're "hard" to train on.

$$\frac{d(A, P)}{\downarrow} \leq \frac{d(A, N)}{\uparrow}$$

$$\frac{d(A, P)}{\downarrow} \propto \frac{d(A, N)}{\uparrow}$$

Schroff et al., 2015, FaceNet: A unified embedding for face recognition and clustering]

- 1 随机取三个的话，loss func 容易满足，训练的识别效果不好
- 2 需要选择难区分的三个，也就是 $D(A, P) D(A, N)$ 比较接近的 a p n，会让算法更努力的学习，训练出的模型更精确！提高 train 效率

[Schroff et al., 2015, FaceNet: A unified embedding for face recognition and clustering]

可以 DOWN LOAD pretrained model

15.5 face-verification-转换为-binary-classification

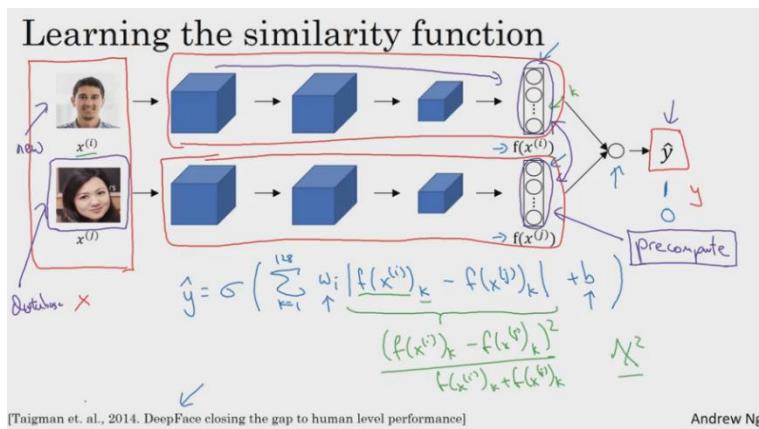
另一种 learn parameter 的方法---logistics，代替 triplet loss
 只需要 1 pair 照片，给 network(也是 siamese-network)
 判断这两张照片是一样(1)，还是不一样(0)

Face verification supervised learning

x	y	
	1	"Same"
	0	"Different"
	0	
	1	

[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

还是先用 deep convnet 来 embedding ! !



pair network 每个照片 compute 2 个 embedding，两个向量对应元素的差的绝对值作为 feature input 向量，给 logistic 进行训练（用 logistics loss function 来 gradient descent）

也可以用卡方(绿色) 卡方相似度 作为 feature

实现：不需要每次将存储照片从新计算一遍得到 output，只需要存储 siamese-network embedding 的结果，来了新的 image 跑一边得到 embedding，直接给 logistics 分类

16 neural-style-transfer 风格迁移

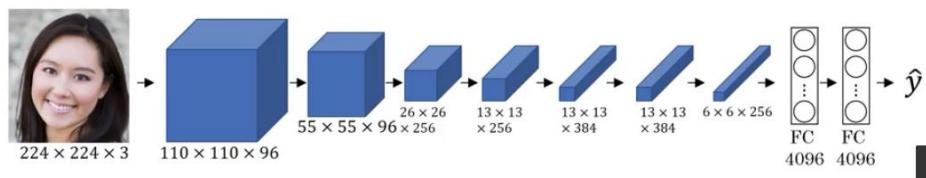
16.1 _what-are-deep-convnets-learning

Visualize: 想知道每个 hidden layer unit really do

[Zeiler and Fergus.. 2013. Visualizing and understanding convolutional networks]

每一层的 unit，通过查看原图对该 unit 的影响，选取原图对该 unit activation 较大的部分，作为该 unit 的作用！！！

Visualizing what a deep network is learning



1 pick a hidden unit in layer 1 (layer 是个三维的 tensor)
2 pass the train to net 看看什么样的图片或图片的一部分 image patch 使得该层 maximize 一个特定的 unit activation (very large activation)
一个 unit 只看到神经网络的很小一部分
选择 9 张 input image max the 一个 unit activation



说明这个 unit 在找斜线

再选择另一个 hidden unit, 是在寻找另一种 pattern (filter 不同)
找颜色, 找形状
每个 unit 都不同的 feature
多做几个, 就可以看出 layer1 在干什么, 一般是很小的 feature

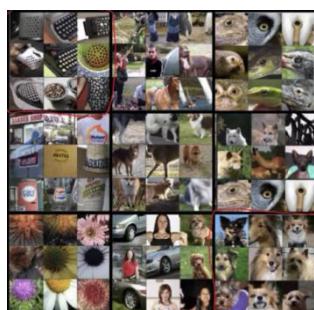
然后再其它层:

Visualizing deep layers



层数越深, 每个 unit feature 的 pattern 越复杂 (复杂的图像), 原来图像更大的 patch

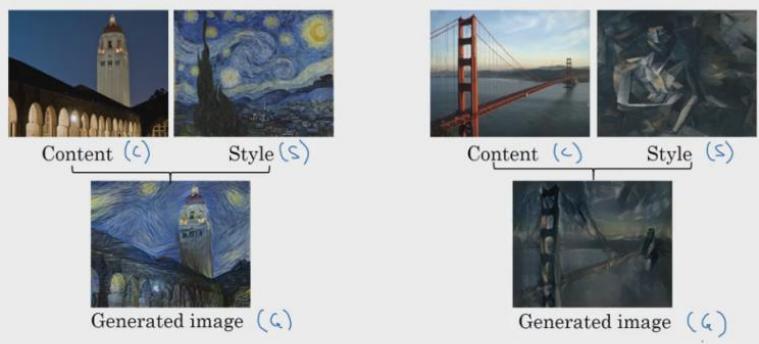
照片的整个结构呈现



可以看出最后一层, 每个 unit 代表什么, 右下角的 unit 代表是狗分类器

16.2 cost-function

Neural style transfer



Images generated by Justin Johnson

Andrew Ng

[\[Gatys et al., 2015. A neural algorithm of artistic style. Images on slide generated by Justin Johnson\]](#) Andrew Ng

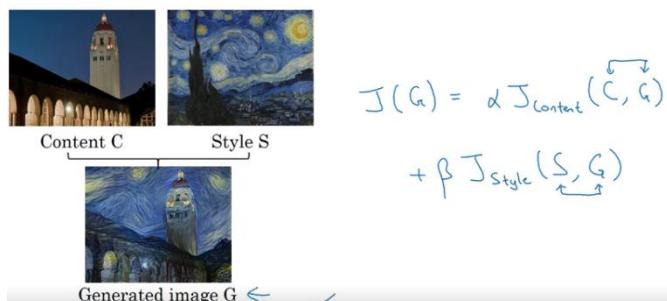
Cost function 衡量 how good 照片的合成，需要两个方面：

$J_{content}$ 保证 C 和 G 的内容 content 不能差别太大

J_{style} 保证 S 和 G 的风格 style 不能差别太大

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

Neural style transfer cost function



1 先随机化 G 照片像素，像素作为参数

2 update the pixel of G , 每个像素的偏导数

Find the generated image G

1. Initiate G randomly

$G: 100 \times 100 \times 3$

2. Use gradient descent to minimize $J(G)$

$$G := G - \frac{\partial}{\partial G} J(G)$$



16.3 content-cost-function

$J_{content}$ 保证 C 和 G 的内容不能差别太大

$J_{content}$ 的计算是在在 C 和 G 选一层 layer (应该是在图像分类的 convnet 里选

一层，因为包含了足够的该图片的特征!!!一般用一个 pretrained 的 convnet, 直接得到 embedding)

layer 不能太 low layer, 因为基本的照片元素都差不多, 也不能太深, 太深的话差别太大, 一般选 middle layer

Content cost function

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

- Say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content

$$J_{content}(C, G) = \frac{1}{2} \|a^{[l](C)} - a^{[l](G)}\|^2$$

[Gatys et al., 2015. A neural algorithm of artistic style]

计算 activation 向量差的来 L_2 norm, 作为 loss func, 距离越大越不相似, 越惩罚!

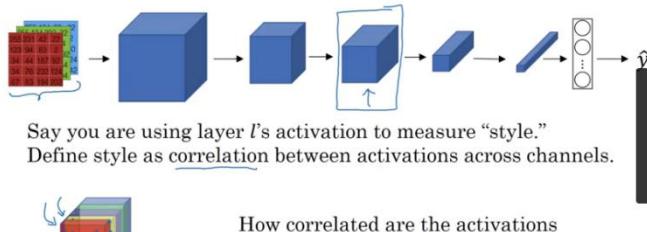
16.4 style-cost-function

还是用一个 hidden layer 来 measure style

define style as correlation between activations across channels

这一 layer 的 风格特征? 就是该层 tensor 每两个 channel 之间的 correlation (一个 channel 一个图片)

Meaning of the “style” of an image



[Gatys et al., 2015. A neural algorithm of artistic style]

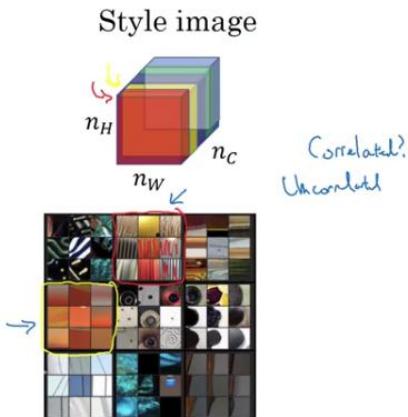
Andrew

假设该 layer tensor shape 是 $nH*nW*nC$

我们知道, 不同 channel 的 activations 的相关性如何?

第一层红色和第二层黄色, 假设 unroll 成一个 vector, 计算对应元素的相关性

如何理解 Correlation? 以及和 style 的关系?



因为每个特定 unit 捕捉原始图片特定的特征
红色和黄色 channel 的相对应上面两个 unit，分别来捕捉两种特征(不同的 texture 质地)

Correlation: 原始图片有红色 unite 特征时，也会有黄色特征(这两个特征会同时出现)的概率

Correlation 告诉我们，每一层 feature，在图片中同时出现的概率，也就是图片的质地

两个 channel 的特征同时出现在这张图片的概率越高，correlation 越高

如何计算图片某一层的风格特征？

首先选取一层 tensor， $a(i, j, k)$ 就是 tensor 的 activation 像素值 (h, w, c)
 k 是 channel

style matrix:

对于 G 和 S image 都要选取一个 layer，计算两个 style matrix :

$G[1][S]$ 和 $G[1][G]$

nc 是 channel 个数，最终得到 $nc \times nc$ 相关系数矩阵 (style matrix)

不同 channel 的相似性！！向量的乘积代表向量的相似度(向量是每个 channel)

Let $a_{i,j,k}^{[l]}$ = activation at (i, j, k)

$a(i, j, k)$ 是 layer tensor 的一个像素值

1 style matrix of G :

先定义 k 和 k' channel 的相关系数，是 style matrix 的一个元素

$$G_{kk'}^{[l](s)} = \sum_{i=1}^w \sum_{j=1}^h a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

同一位置(i, j) 像素值的乘积，的和

同样对于 G image:

$$G^{[l]}(G) = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

上面的矩阵也叫(格拉姆矩阵/Gram matrix)

Gram matrix G of a set of vectors (v_1, \dots, v_n) is the matrix of dot products
给任意一个 image 可以计算一个 style matrix

可以看出当两个对应的 activation 都大时，correlation 值(乘积)越大

其实值是: the unnormalized cross covariance 交叉协方差 because we're not subtracting out the mean

3 定义 style cost

objective: 保证 S 和 G 的风格不能差别太大，也就是两个 style matrix 接近
其实将距离作为 cost 就好！！！

$$\begin{aligned} J_{style}^{[l]}(S, G) &= \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \| G^{[l](S)} - G^{[l](G)} \|_F^2 \\ &= \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})^2 \end{aligned}$$

$$J_{style}^{[l]}(S, G) = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})$$

对于 S, G image 某一隐层 l 的 cost

两个 style matrix elementwise Frobenius norm 模，加上标准化因子
(2*nw*nh*nc) 平方

4 J_{style} 的最终版本

如果我们用多个不同的 layer 来要求 style cost 效果会更好！！！

多个 layer 的 sum, 每个 layer 有个权重 lambda

$$J_{style}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S, G)$$

5 最后总的 cost func:

$$\underline{J(G)} = \alpha J_{content}(G) + \beta J_{style}(S, G)$$

只需找到 G , minimize J

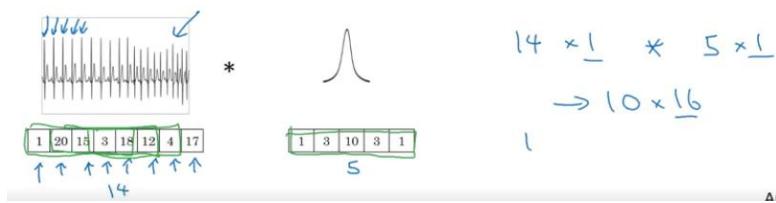
17 1d and 3d tensor generalizations

我们通常说的 2D, 是不包含 channel 的!! 因此图片属于 2D!

ConvNets from 2D data to also 1D as well as 3D data

其他维度 data:

1d: 时间序列数据



时间序列数据长度为 14 shape 14*1 向量
filter shape 5*1 16 个 filter
output: 10*16 tensor

下一层继续 长度是 5 的 filter (5*16), 32 个 filter 假设 32 个 filter

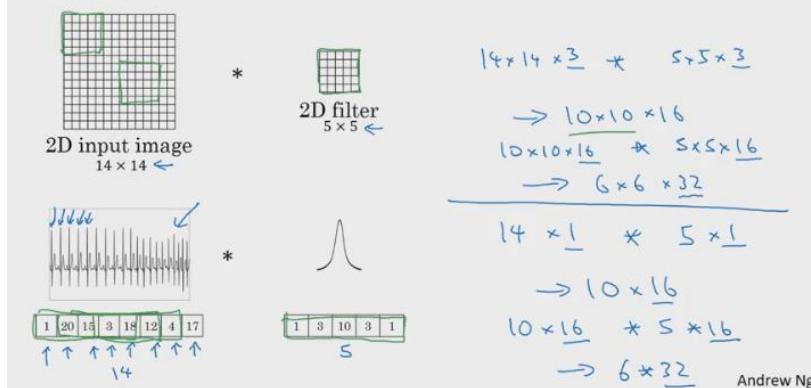
$$10 \times \underline{16} * \underline{5 \times 16} \\ \rightarrow 6 \times \underline{32}$$

Andrew Ng

output: 6*32 tensor

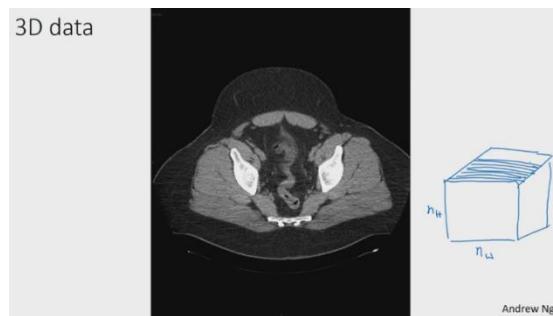
对比:

Convolutions in 2D and 1D

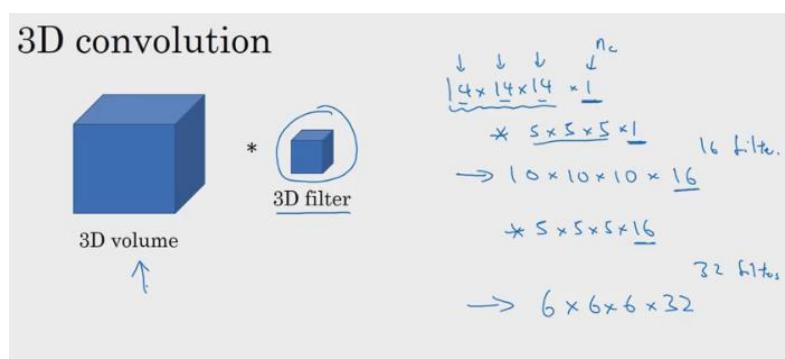


1d data 更多用在 **rnn** 上

3d: 核磁共振 CAT scans, medical scans



3D convolution



图片 channel 为 1 黑白的
filter $5 \times 5 \times 5 \times 1$ 共 16 个 filter, 得到 $10 \times 10 \times 10 \times 16$

movie data 也是 3d

一段 movie slices 就是个 3D 数据

$100 \times 200 \times 30 \times 3$

假设 1 分钟有三十帧图片, 3 个 channel

Filter 是为了 detect feature in the data! ! !

recurrent neural networks

1 recurrent neural networks

1.1 intition 和传统神经网络局限

input and output 都是 word sequence, 也可以只有一个 is word
Examples of sequence data

Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation		→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCTGTGAGGAAC TAG	→	AGCCCCTGTGAGGAAC TAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger. Andrew Ng

input 和 output 也可以长度不相等! !

Name entity

Motivating example

x: (Harry Potter) and (Hermione Granger) invented a new spell.
 $\begin{matrix} x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(T_x)} & \dots & x^{(q)} \end{matrix}$
 $T_x = 9$

→ y: l l o | | 0 0 0 y⁽¹⁾ y⁽²⁾ y⁽³⁾ ... y^(T_y)
 $T_y = 9$

$\begin{matrix} x^{(i)} & T_x^{(i)} \\ y^{(i)} & T_y^{(i)} \end{matrix}$

x 是一句话, 每个 feature 一个 words

y label sequence

T 代表 len of words

$\langle t \rangle$ 表示 feature () 表示第几个 example, 第几句话
每个 example T 长度不同

Represent word 需要将 word 编码成计算机用的

1 需要一个 vocabulary: 我们需要用到的所有词汇 bag

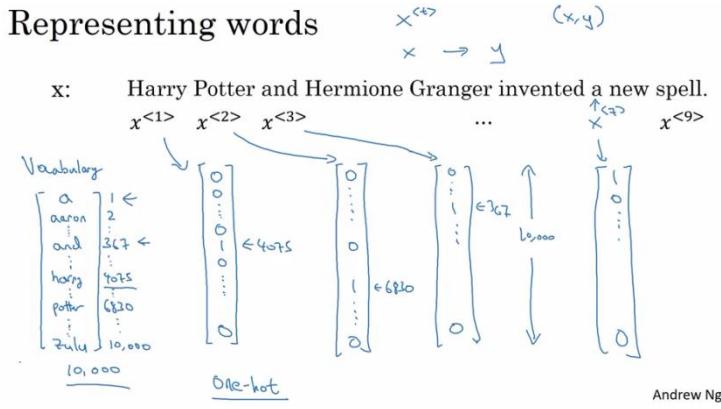
2 每个 word 对应一个 index

一般商业应用的词汇量在: 3-5 万

word 表示: 用 onehot-encoding vector 表示

y 需要我们自己先标记 0, 1

然后 train sequence model



然后一个 x 就是将 9 个 word 拼起来成为一个超级长的 vector，给神经网络
传统神经网络缺点：

1 每个 example, 每句话 word 数量不同, 而且 input vector 太长了

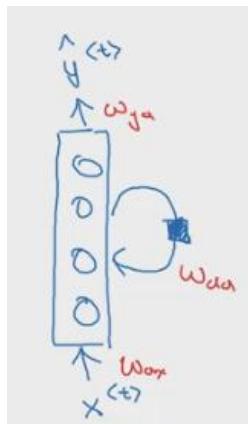
2 不 share feature learned across 不同的 position of text

假如从 input 第 i 个位置学到 $harry$ 是个 name, 不能 generalize to 另一个位
置(另一个位置看到 $harry$ 也不认识), 需要从新学习, 不会共享参数

不像 convnet ! !

1.2 recurrent-neural-network-model

Recurrent-neural-network representation



定义一个网络结构：input (X) ----hidden layer (a) ----output layer (y)

但每个 time step t 都运行这同一个网络, 但上一次的中间结果输出, 会给下
一次。因此所有 time step 公用一套 parameter(同一个神经网络)

1 所有 words 公用一套神经网络参数 **share parameter (identical)**

2 t 时刻的 word 会用到 $t-1$ 时刻的信息($t-1$ 时刻 input 信息), 通过 activation
传递, 前面时间的信息会传递给后面时间

具体细节：

W_{xy} 是矩阵 $y \rightarrow x$ 的参数

1 t 时刻中间隐层的 activation a 会做全连接变换(fully connect 乘以一个矩阵)并被传递且 add 到下一层 $t+1$ 的 z (元素相加)

$$a^{<t+1>} = g(W_{aa}a^{<t>} + z^{<t+1>})$$

$$a^{<t+1>} = g(W_{aa}a^{<t>} + W_{ax}x^{<t+1>} + b_a) \quad \text{隐层 activation 一般是 tanh/relu}$$

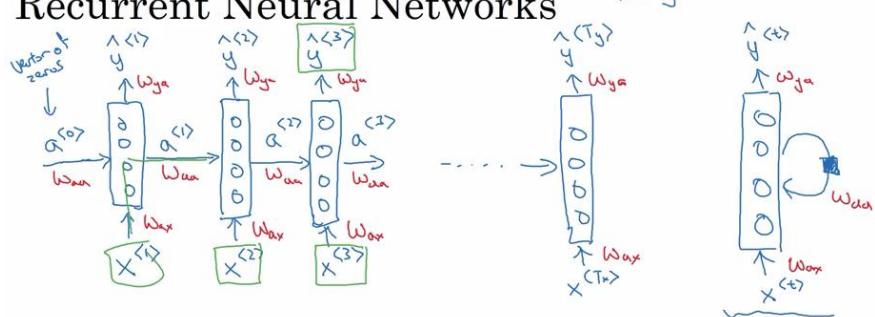
W_{aa} 是前一层的隐层 a , 到下一层的 a

2 然后当前层的 activation 会产生 y

$$y^{<t+1>} = g(W_{ya}a^{<t+1>} + b_y) \quad \text{output 层 activation 一般是 sigmoid(分类)}$$

下面是展开的形式：更容易理解，但要记住，其实只有一个网络！！！

Recurrent Neural Networks



每一个 time step t 的 y 输出，会用到了前面所有网络输出的信息，只用到前面的信息

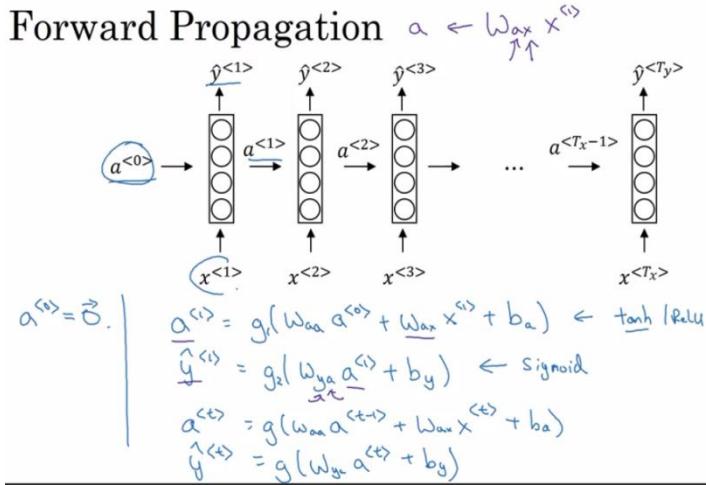
这也是缺点！！例如：

He said, “Teddy Roosevelt was a great President.”

He said, “Teddy bears are on sale!”

我们不能根据前面的 word 来预测 Teddy 是否为人名，需要根据后面的信息推测

Forward propagation



a^0 作为一个初始向量

$a^{<t>}$ 和 $x^{<t-1>}$ 公用一个 bias, 因为 $a^{<t>}$ 和 $x^{<t-1>}$ 都可以看做 input 接一个全连接

计算 activation 的激活函数, 经常用 tanh/relu

计算 y 的激活函数, 经常用 sigmoid, 或 softmax

simplify 公式:

Simplified RNN notation

$$\begin{aligned} a^{<t>} &= g(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a) & a^{<t>} &= g(W_a [a^{<t-1>} \mid x^{<t>}] + b_a) \\ y^{<t>} &= g(W_{ya} a^{<t>} + b_y) & \uparrow \begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix} &= W_a \\ & & & (100, 10100) \end{aligned}$$

$$[a^{<t-1>} \mid x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \quad \begin{array}{c} \uparrow 100 \\ \uparrow 10100 \\ \downarrow 10100 \end{array}$$

将 weight 合并 (100, 10100)

$a^{(t-1)}$ 和 $x^{(t)}$ 只需 stack 成一个大向量 (10100, 1)

两个分块矩阵相乘, 得到原来的式子 (100, 1)

y 也简化形式:

$$\begin{aligned} \hat{y}^{<t>} &= g(W_{ya} a^{<t>} + b_y) \\ \hat{y}^{<t>} &= g(W_y a^{<t>} + b_y) \end{aligned}$$

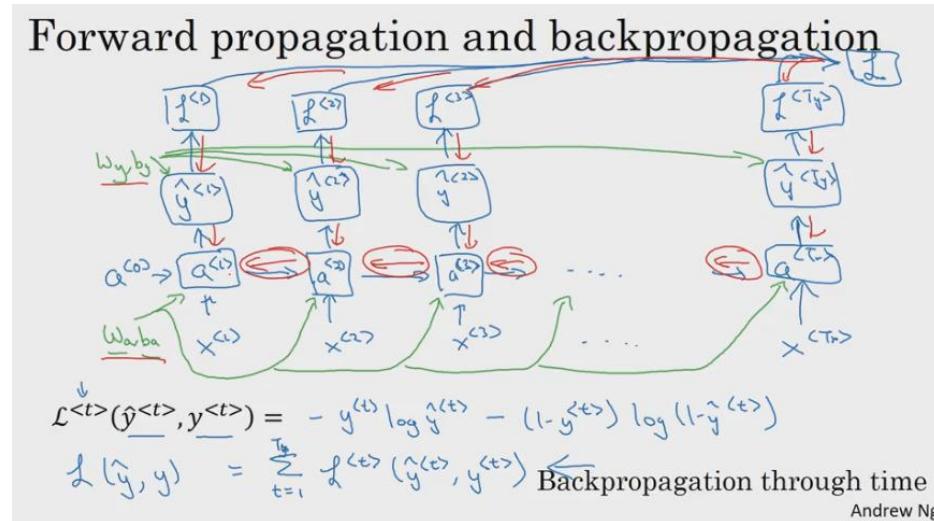
这样所有 W 的下标表示 output 的类型!!!

backpropagation-through-time (question 如何 backpropagation)

所有时刻， w_a b_a w_y w_b 都是同一套参数

对于一个 t 时刻的输出 y ，用 cross entropy 作为 cost func

总的 Cost : 所有的时刻 t 的输出 cost func 加总

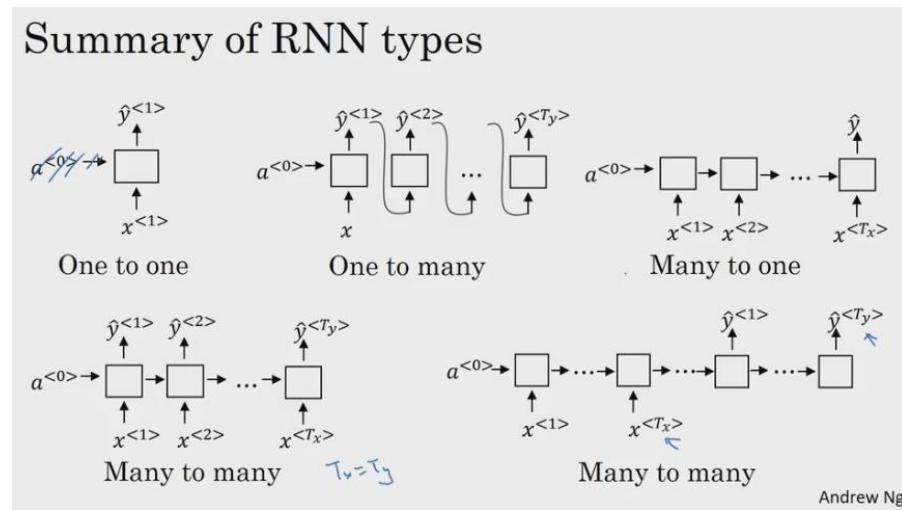


中间的 w_a b_a 的后向传播被称为 back pro through time

当做一个 deep net，每一层一个 time step，进行后向传播！！！跟前向传播的过程刚好相反

different-types-of-rnns

Andrej Karpathy



1 many to many 结构: T_x T_y 相同 input 数量和 output 数量相同

2 many to one 情感分析: sentiment classification binary

y: 离散的情感 state

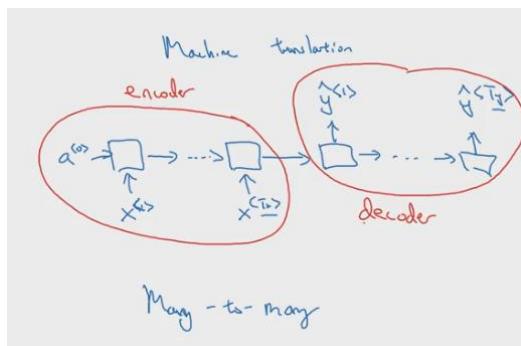
3 one to many

music generation

给定一个指令，生成音乐

4 many to many 结构: Tx Ty 不同

translate:



encoder decoder

1.3 language-model-and-sequence-generation

speech recognition: 语音识别，语音 \rightarrow text

核心部分是：language model 作用：给出 sentence 的概率

给定一个 sentence(sequence)，得出该 sentence 出现概率

$$P(\text{sentence}) = ? \quad P(y^{(1)}, y^{(2)}, \dots, y^{(T)})$$

tokenize the sentence 词表示

Language modelling with an RNN

Training set: large corpus of english text.

Tokenize

Cats average 15 hours of sleep a day. \downarrow $\langle \text{EOS} \rangle$
 $y^{(1)} \quad y^{(2)} \quad y^{(3)} \quad \dots \quad y^{(10)} \quad y^{(11)}$

The Egyptian Mau is a bread of cat. $\langle \text{EOS} \rangle$
 $\langle \text{UNK} \rangle$

1 map each word to onehot vector(individual token 记号, 代表)(每个 word 在 vocabulary 的 indice)

2 增加 EOS token 代表每个句子结束

上面一共 9 个 input

也可以将标点符号加入 vocabulary 里面。

如果 sentence 一些 word, 不在 vocabulary 中, 用 UNK token 代替 unknown word

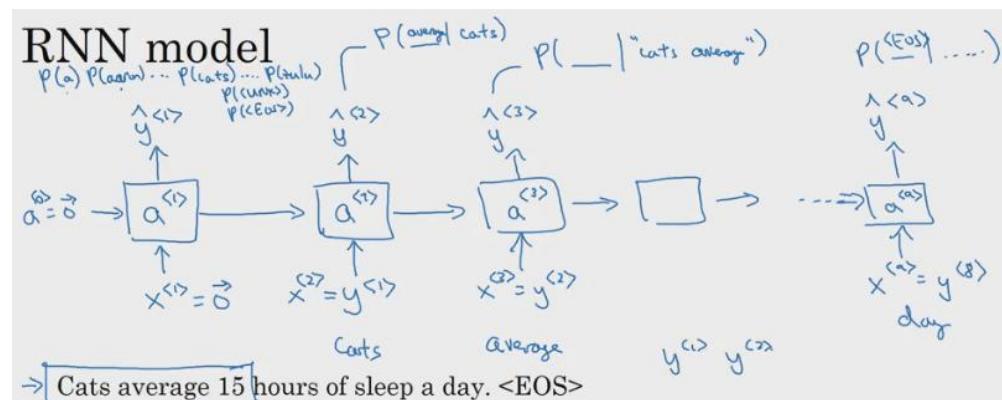
UNK 和 EOS 都在 vocabulary 里

x 是一个 word 的 onehot vector

y 是 softmax 概率值 (vocabulary 里所有词汇的概率 vector)

y 是长度为 $\text{len}(\text{vocabulary})$ 的 fully connect layer 的 softmax

例：



x, a 初始化为 0 向量

$y \hat{1}$ 就是 softmax 概率值输出 (所有 vocabulary 的 word 的概率向量), 根据这个概率向量进行 sample, 或取最大值, 假如是 cat

得到 cat 的真实 label onehot y_1 作为下一次的输入 X

$x_2 = y_1$ (注意不是 y_{ba})

$y \hat{2}$ softmax 输出的概率是 $p(\text{average/cat})$

每个 $y \hat{}$ predict 是: $p(\text{next word} | \text{previous words})$

cost func:

目的是让输出节点来推测下一个 word 出现的概率, 通过训练让正确下个 word 概率最大化 ! !

先单独看每个 loss i 是每个 vector 的元素 (softmax loss)

所有时刻 t loss sum 就是 cost

$$\begin{aligned} \mathcal{L}(\hat{y}^{(t)}, y^{(t)}) &= - \sum_i y_i^{(t)} \log \hat{y}_i^{(t)} \\ \mathcal{L} &= \sum_t \mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)}) \end{aligned}$$

让网络学会我们的用词规律

可以完成功能:

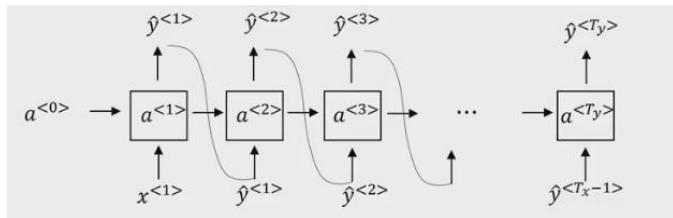
1 给定任意一个 word, predict 下个 word 出现的概率 ! !

2 给定一句话, 给出这句话的概率: 其实也是重复的将这句话中每个 word, 1 个 1 个的 input 到这个网络, 找到我们要的 output 概率值, 然后相乘得到概率 !!

$p(w_1, w_2, w_3) = p(w_1) * p(w_2 | w_1) * p(w_3 | w_1, w_2) = y \hat{1} * y \hat{2} * y \hat{3}$

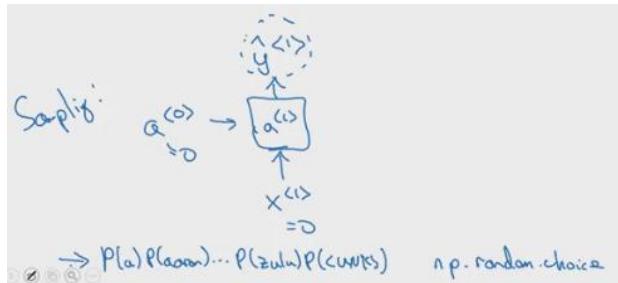
1.4 sampling-novel-sequences (hmm 和 RNN)

sample: 上一步预测了什么，就作为下个的 input (HMM 的原理)



a^0, x^1 初始 0 vector 得到 y^1 softmax 概率

sample:



得到 t 时刻的 softmax，其实是个 distribution，每个人的概率，用这个概率来随机抽取一个名字，作为下一个 t+1 的 input

`np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0])`

从 range(5) 里按照 [0.1, 0, 0.3, 0.6, 0] 来 随机选取！！！

$x(t)=y(t-1)$

RNN HMM

应用的在序列问题

都是假设前面的 node 会影响后面的 node

HMM 只假设，当前 node 只受前一个 node 影响

RNN 假设，当前 node 只受前面好几个 node 影响，而且可以很久很久！！

例如：pet dog whose mood is heavily dependent on the current and past few days' weather.

You've collected data for the past 365 days on the weather, which you represent as a sequence as $x^{<1>} \dots x^{<365>}$. You've also collected data on your dog's mood, which you represent as $y^{<1>} \dots y^{<365>}$. You'd like to build a model to map from $x \rightarrow y$ by using RNN

1.5 Character level language model RNN

以上是 Word level language model RNN

Vocabulary: 是字符

$$\text{Vocabulary} = [a, b, c, \dots, z, \cup, ., , ;, 0, \dots, 9, A, \dots, Z]$$

也可以应用在 NLP

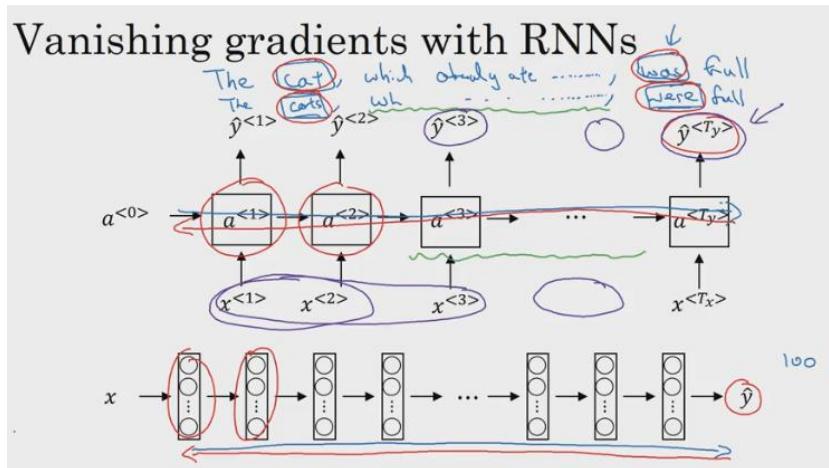
只是每个字符作为 input

好处：消除 unknown token 不会 vocabulary 不存在

坏处：每句话 char 太多了！！train 计算量太大，sequence 太长！！

1.6 vanishing-gradients-with-rnns (gradient clipping)

句子中间有插入语或从句时：cat 和系动词相关联，但跨度太大，需要更长记忆



long-term dependency(长周期依赖)：需要记住很久以前的网络的 input 值，可以影响未来的网络的 output

但 **basic RNN** is hard to memorize 前面网络的值，Basic RNN 只受前面几个 close 网络 input 的影响

原因---Vanishing gradient: for very deep neuron network ,后向传播很难传到前层，RNN 同样也是。

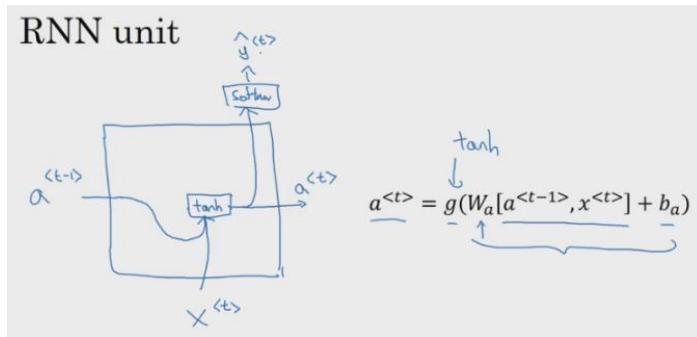
Exploding gradient: 使得 parameter 巨大，也会影响 RNN

当看到 parameter 是 NaN 或 not a number Exploding gradient

solution: **gradient clipping**, look gradient vectors, 如果 bigger than some specific threshold, re-scale some gradient vector

1.7 gated-recurrent-unit GRU (RNN 的变形)

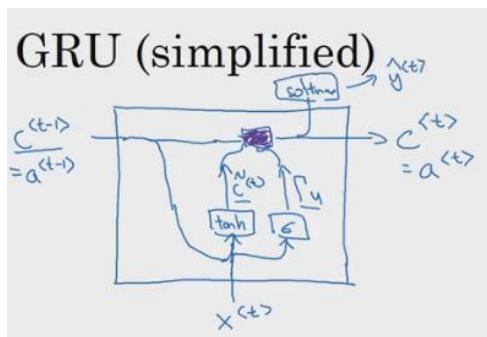
1 Long dependency 2 decline the Vanishing gradient



Gated-recurrent-unit

→ The cat, which already ate ..., was full.

[Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches] ←
 [Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling] ←



没有 a 了，将 a 改成 c 了！！

Memory cell (C) : remember 记住 cat 是单数还是复数
C 本质是之前的 activation, merely change the name

1 C tilde:

$$\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

C tilde of t candidate of C(对 C 值的更新)

activ-func: tanh

2 Gate: Gate control whether to update title C to C

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

gama U (U:update) (因为 gama 是 G 的大写)

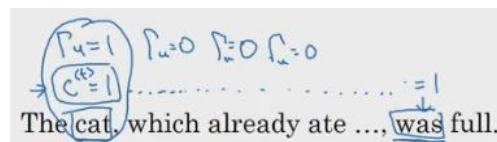
activ-func: sigmoid sigmoid 更多的值的分布是接近 1 或 0 (gate 值)

3 1,2 是用来计算当前 t 的 C

C 传递到两个地方：1 是 hidden layer 2 紫色部分需要和 C tilde 相加
gama U 和 tilde 相乘，

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) + c^{<t-1>}$$

EXAMPLE:



第一次遇到 cat 时，会设置 $C=1$ ，一直 remember(记录单复数形式)，一直到碰到系动词还是 $C=1$ 。

而 Gama 每个元素决定每个 word 的 $C^{<t>}$ 是 0 还是 1，是否 update 成 1
单数 set=1 复数 set=0

我们希望中间部分 gama 全部 $G=0$ no update

上图紫色部分是 update 公式：

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) + c^{<t-1>}$$

最后对输出的 update，* 是 element wise 相乘

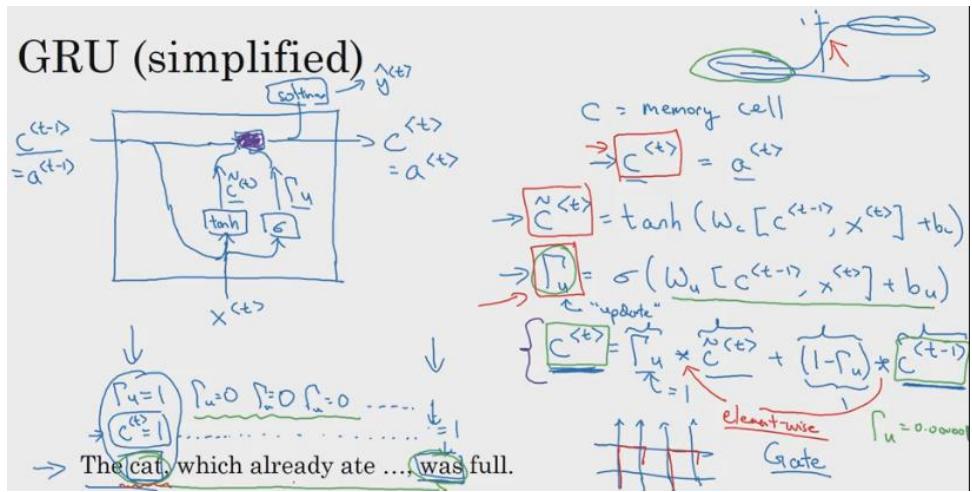
G 也是一个 vector，元素大多是 0 或 1，只是起到一个 gate(开关)作用，vector
元素为 1 就表示 update，元素为 0 就不更新，是对 $c^{<t>}$ (也就是前面的 activation
值)每个元素的更新

可以用 gate 的一个元素来 remember，是否单复数
另一个元素来 remember，是否是 food

三个向量： $C^{<t>}$, $C \text{ tild}^{<t>}$, gama 的 dimension 都一样！！！

好处： G 的是经过 sigmoid，值大多是 0,1， $G(gama)$ 容易设置成 0，只要 z 是负数。此时 $C^{<t>}$ 几乎等于 $C^{<t-1>}$, value of $C^{<t>}$ 一直 maintain(无论多少 step)
question? ? 解决了 Vanishing gradient，不能 long dependency 问题
当前网络，可以仍然可以得到很久之前网络的 activation 值！！！

summary:



以上是简化公式，GRU 最终的公式：

Full GRU

$$\begin{aligned}\tilde{c}^{t-1} &= \tanh(W_c [\Gamma_r * c^{t-1}, x^{t-1}] + b_c) \\ \Gamma_u &= \sigma(W_u [c^{t-1}, x^{t-1}] + b_u) \\ \Gamma_r &= \sigma(W_r [c^{t-1}, x^{t-1}] + b_r)\end{aligned}$$

再加个 Gate r

Gama r (relevant 相关性) how relevant C^{t-1} to $C \tilde{c}^{t-1}$

Gama r 计和 Gama u 计算一样，只是 parameter matrix 不同：Wr br
有些文献会用到其他符号表示：

Full GRU

$$\begin{aligned}\tilde{c}^{t-1} &= \tanh(W_c [\Gamma_r * c^{t-1}, x^{t-1}] + b_c) \\ \Gamma_u &= \sigma(W_u [c^{t-1}, x^{t-1}] + b_u) \\ \Gamma_r &= \sigma(W_r [c^{t-1}, x^{t-1}] + b_r) \\ c^{t-1} &= \Gamma_u * \tilde{c}^{t-1} + (1 - \Gamma_u) * c^{t-1}\end{aligned}$$

GRU 公式总结：

$$\begin{aligned}\tilde{c}^{t-1} &= \tanh(W_c [\Gamma_r * c^{t-1}, x^{t-1}] + b_c) \\ \Gamma_u &= \sigma(W_u [c^{t-1}, x^{t-1}] + b_u) \\ \Gamma_r &= \sigma(W_r [c^{t-1}, x^{t-1}] + b_r) \\ c^{t-1} &= \Gamma_u * \tilde{c}^{t-1} + (1 - \Gamma_u) * c^{t-1} \\ a^{t-1} &= c^{t-1}\end{aligned}$$

1.8 long-short-term-memory-lstm (一定看作业)

(比 GRU 更好)记住更长的内容

[Hochreiter & Schmidhuber 1997. Long short-term memory]

hidden-state a^{t-1} and cell-states c^{t-1}

每单元会输出 a^{t-1} , c^{t-1} , 是独立的两个值, 都是 hidden 的值, a 是激活后, c 未激活(c 相当于 z)

a^{t-1} 只和 x^{t-1} 在一起计算, 计算 gate 和当前 c tilde 值

c^{t-1} 只用来计算 c^t , 只用来遗忘

c tilde 用来 update

1 C tilde (candidate 记忆值)

2 三个 independent update Gate (不再是 G 和 $1-G$) 3 个:

update Gate + forget Gate + output Gate

forget Gate 控制 c^{t-1} , 是否忘记过去

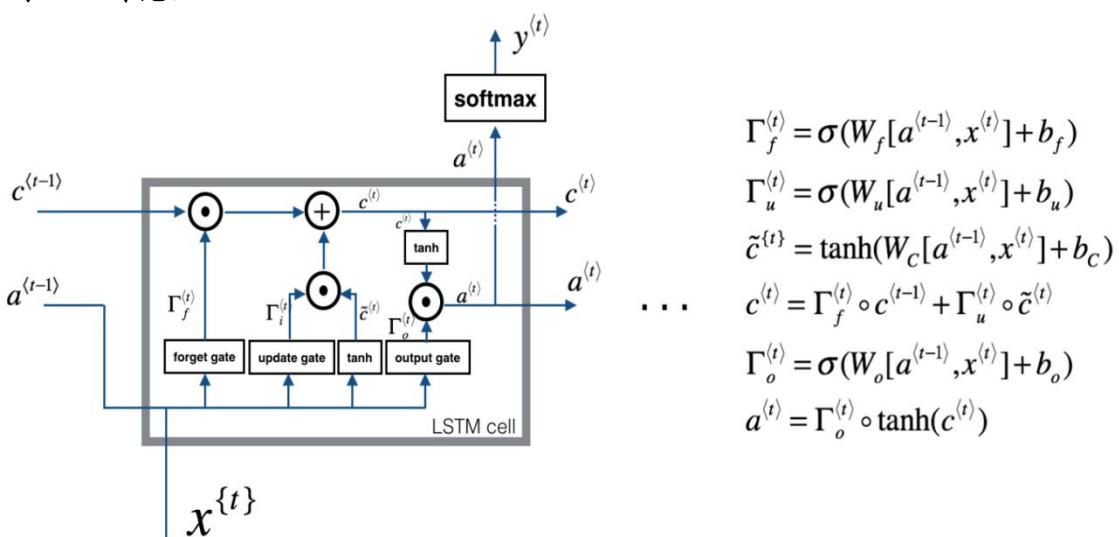
update Gate 控制 c tilde, 是否更新现在

3 update equation: 更新 C^t 用独立的两个 Gate, 最终得到 c^t

4 $\tanh(c^t)$ 得到激活值 a , 再用 output Gate 筛选, 得到最终 a^{t-1}

5 a^{t-1} 作为输出, 也作为计算 y hat 的 hidden 值!

与 GRU 对比:



1 每个 unit 三个 input: c^{t-1} , a^{t-1} , x^t

2 三个 Gate vector 维度一样!!! 和 hidden layer 的数量一样 (activation 数量), 计算方式也一样, input 一样 (a^{t-1} , x^t), 只是参数不同, 功能一样都是 gate

3 C tilde 也是用 a^{t-1} 和 x^t 来计算

所有 gate 和 c tilde 都是同一个 input, 只是参数和激活函数不同！！

4 计算 c

* 代表 elements 乘

+ 代表 相加

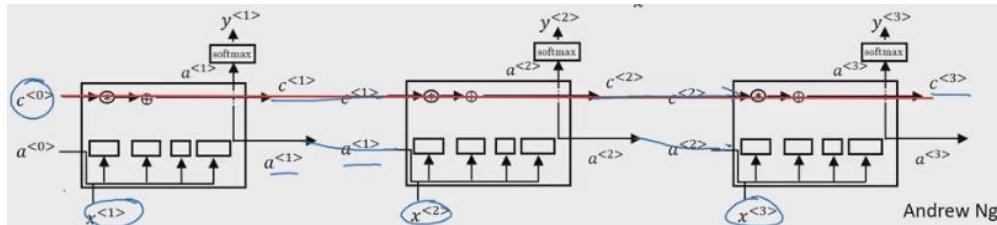
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

只要设置好 update Gate 和 Forget Gate, 就可以很好的 maintain $C^{<t-1>}$ 的某些 element 信息, 一直不变传递到后面时刻网络

5 得到激活值 a 后还要乘以 output gate!! 最终激活值 a

6 得到的最终激活值, 1 是和 c 值一起传递给下一层, 2 是用于计算该层的 y (fully connection softmax)

多个单元连接起来, 就是循环神经网络



Andrew Ng

variation:

$$\begin{aligned} \Gamma_u &= \sigma(W_u [a^{<t-1>} x^{<t>}] + b_u) \\ \Gamma_f &= \sigma(W_f [a^{<t-1>} x^{<t>}] + b_f) \\ \Gamma_o &= \sigma(W_o [a^{<t-1>} x^{<t>}] + b_o) \\ c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\ a^{<t>} &= \Gamma_o * \tanh c^{<t>} \end{aligned}$$

peephole connection

计算 Gate 时, 可能会在里面多一个 $c^{<t-1>}$ (也传递进来) peephole connection
注意一点: 每一个 gate 的元素, 只受同样该位置的 $a^{<t-1>}, x^{<t>}, c^{<t-1>}$ 元素的影响!!!

GRU 更新出现, 简单, 计算量小, 可以 build 更深的网络
LSTM more powerful, 是 default 首选!

1.9 bidirectional-rnn BRNN

He said, "Teddy bears are on sale!"

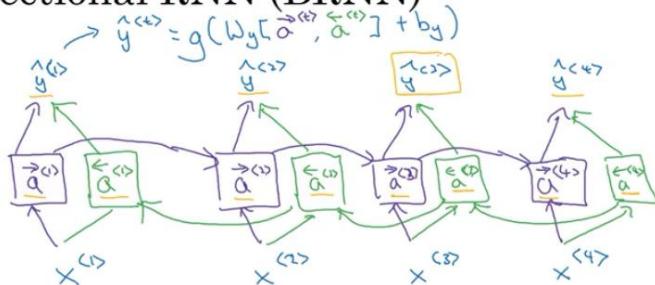
He said, "Teddy Roosevelt was a great President!"

可以从后面的节点获取信息

add backward recurrent layer, 呈现对称的结构

首先 backward recurrent layer 会相互后向的连接

Bidirectional RNN (BRNN)



forward 方向计算 a forward

backward 方向也计算 a backward

然后合并成一个向量，来预测 y, 上图 y 的计算

Acyclic graph

input forward pass; compute a forward 1, a forward 2...

input backward pass: a backward 1, a backward 2...

y hat₃ 输出，不仅受前面网络 input x 的影响，也受后面网络 input x 影响。

因此后面的 input word 也会传递到前面

最佳实践: bidirectional + LSTM

缺点: predict 时需要用到全部的 word 的信息，完整的句子，因为每个 y hat 计算都用到全部的 input，都要全部计算 forward 一遍

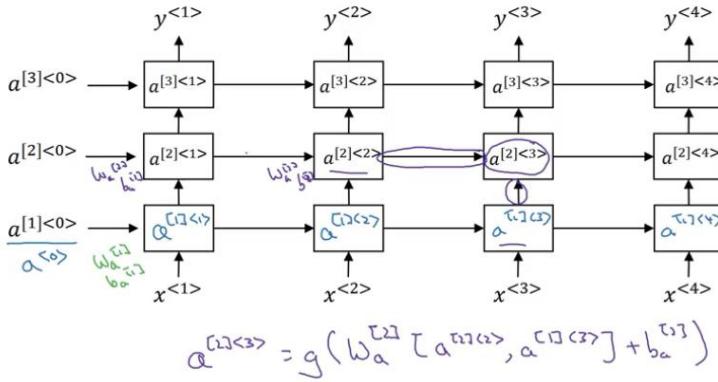
语音识别，需要等全部说完话，才能预测，不适合 realtime

1. 10 deep-rnns

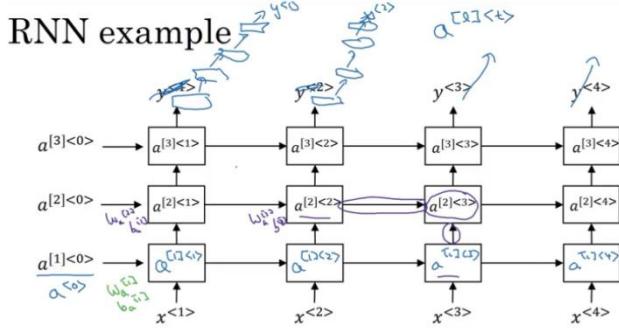
3 层已经算是深了(因为循环次数多)，多个 unit stack 相当于多个 hidden layer stack，每一个 hidden layer 的激活值或 z，保留并传递给下一个 t 时刻的同一层！！

每个 unit 的激活值 activation 给作为下一个 unit 的 x

[i] 代表第 i 层，每一层 share parameter



最后一层可以继续接 deep net，可以在那里做深



1.11 coding 作业(rnn 的独特维度)和 loss 计算

RNN 只需创建一个网络结构，每次 step 传递一个 X<t>

正常 X(m,T,n)

但有一种奇怪的 shape，为了将每个 word 组织成向量

X<t>: (n_x,m) 一个 timestep 的（第 t 个 word）的 shape

顺序是为了把每个 example 的特征变成一个列向量！

对于多个 Time step，维度有点奇怪！！

X: (n_x, m, T)

如：(vocabulary len=3, m=2) 每一列是特征 onehot(假装是)，m 句话
 $[[-0.3224172 \quad -0.38405435]]$
 $[\quad 1.13376944 \quad -1.09989127]$
 $[\quad -0.17242821 \quad -0.87785842]]$

(voca=3, m=2, Tx=2) 每句话的长度是 2， 2 个向量，每句话 2 个 onehot
 之前的每个 onehot 元素扩展成长度为 2 的数组，数组就是句子长度 Tx
 $[[[\quad 0.21202593 \quad -0.75291473]]]$

$[-0.53702145 \quad 0.46865883]$

$\begin{bmatrix} [1.49382447 \quad -0.08998659] \\ [0.83148745 \quad -2.05193068] \end{bmatrix}$

$\begin{bmatrix} [-0.89321467 \quad 0.31370808] \\ [-0.08437715 \quad -1.89521715] \end{bmatrix}$

最外层三个数组，代表 voca
里面的两个子数组代表 m
子数组的每个元素是第几个 word

切片 $[:, :, 0]$

就是所有 voca, 所有 m, 的第一个 word (3, 2)

$\begin{bmatrix} [0.21202593 \quad -0.53702145] \\ [1.49382447 \quad 0.83148745] \\ [-0.89321467 \quad -0.08437715] \end{bmatrix}$

一个 example: (voca_len, Tx), 也就是一个计算 loss 的单位

2 natural-language-processing-word-embeddings

2.1 introduction-to-word-embeddings

word representation NLP and Word Embeddings

Word representation

$V = [a, aaron, \dots, zulu, <UNK>]$

1-hot representation

Man	Woman	King	Queen	Apple	Orange
(5391)	(9853)	(4914)	(7157)	(456)	(6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$
$\textcircled{0}_{5391}$	$\textcircled{0}_{9853}$				

V 是 vocabulary

0 代表 Onehot encoding

缺点：只能代表这个 word 的本身，没办法跨越 word，没办法推断 word 之间关系 similarity

I want a glass of orange juice.

I want a glass of apple ?.

onehot 两个 word 直接没有什么关联，如理应 apple 和 orange 都属于水果，应该更接近，学了上面一句，会推出下面也是 juice，但实际上 apple 和 orange 并没有关联

因为：任何两个向量的乘积是 0，没办法定义 word 直接的距离

Word embedding: 用少的几个 feature 代替 high dimension onehot, represent 每个 word

Featurized representation: word embedding

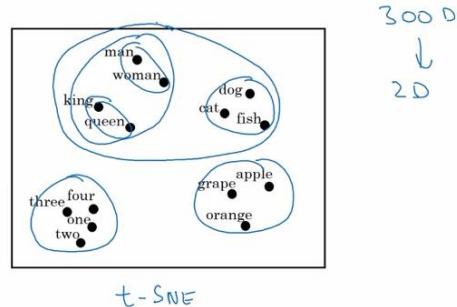
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
size						

很多 feature，每个 vector 用 e(embedding) 来表示
apple 和 orange feature vector e 很相似，遇到了都给 juice

t-SNE algorithm 可视化

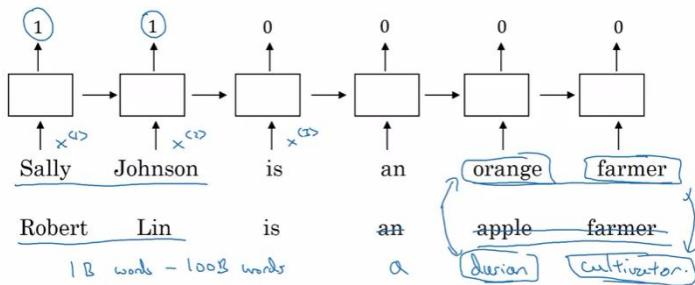
将高维 e vector 降成 2 维，可视化
可以看出每个 word 的距离

Visualizing word embeddings



using-word-embeddings (trasfer learning)

Named entity recognition example



1 经过大量的 unlabel 数据训练，可以找到 word 直接的 similarity，learn word 的 embedding 向量，

2 再用有 label 的 small dataset，feature 就是 embedding 进行训练用 bidirection RNN，从而可以应用在 named entity (一句话里人名的识别)，

其实也是一种 transfer learning

A 数据很多，B 数据很少：用 A 训练好特征，用 B 做分类(new task)。

Transfer learning and word embeddings

1. Learn word embeddings from large text corpus. (1-100B words)

(Or download pre-trained embedding online.)

2. Transfer embedding to new task with smaller training set.
(say, 100k words) $\rightarrow 10,000 \rightarrow 300$

但 Embedding vector dimension 远小于 onehot 维度

embedding 和 face encoding 是一回事！！！

face encoding

将 face 通过训练得到输出向量 encoding

properties-of-word-embedding

embedding 的性质：更好的理解 embedding

1 embeding 学到了 analogy 类比

[Mikolov et. al., 2013, Linguistic regularities in continuous space word representations]

下面两个 embedding 的差，应该相等

e man-e woman 得到新的向量 1

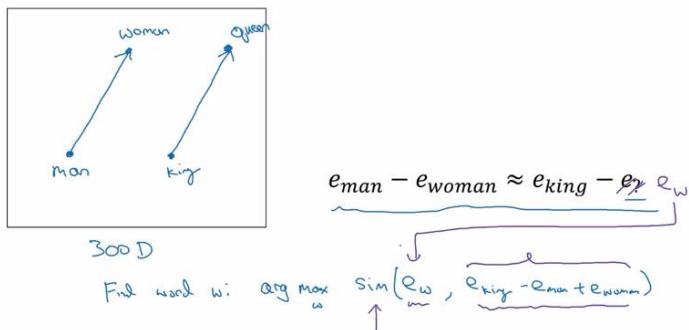
e king-e queen 得到新的向量 2

因为 main difference 是 gender

可以用来做 analogy reason 类别推断

man-woman 等价于 king- ? 找到最接近的 word

Analogies using word vectors



只需 find the word w , $e_{\text{king}} - e_w$ 等于 $e_{\text{man}} - e_{\text{woman}}$

sim(向量, 向量)两向量的相似度

cos similarity 或 square dist(disimilarity)

两个向量的夹角的 cos, 夹角越大越不相似, cos 越接近-1(180 度), 越相似为 1

Cosine similarity

$$\rightarrow \boxed{\text{sim}(e_w, e_{\text{king}} - e_{\text{man}} + e_{\text{woman}})}$$

$$\text{Sim}(u, v) = \frac{u^T v}{\|u\| \|v\|}$$

Man:Woman as Boy:Girl
Ottawa:Canada as Nairobi:Kenya
Big:Bigger as Tall:Taller
Yen:Japan as Ruble:Russia

分子; inner product, 分母标准化

结果是两个向量的 cos angel

角度越大 越不相似

embedding 学到的 pattern

Man:Woman as Boy:Girl

Ottawa:Canada as Nairobi:Kenya

Big:Bigger as Tall:Taller

Yen:Japan as Ruble:Russia

Which of these equations do you think should hold for a good word embedding? (Check all that apply)

$e_{boy} - e_{girl} \approx e_{brother} - e_{sister}$

$e_{boy} - e_{girl} \approx e_{sister} - e_{brother}$

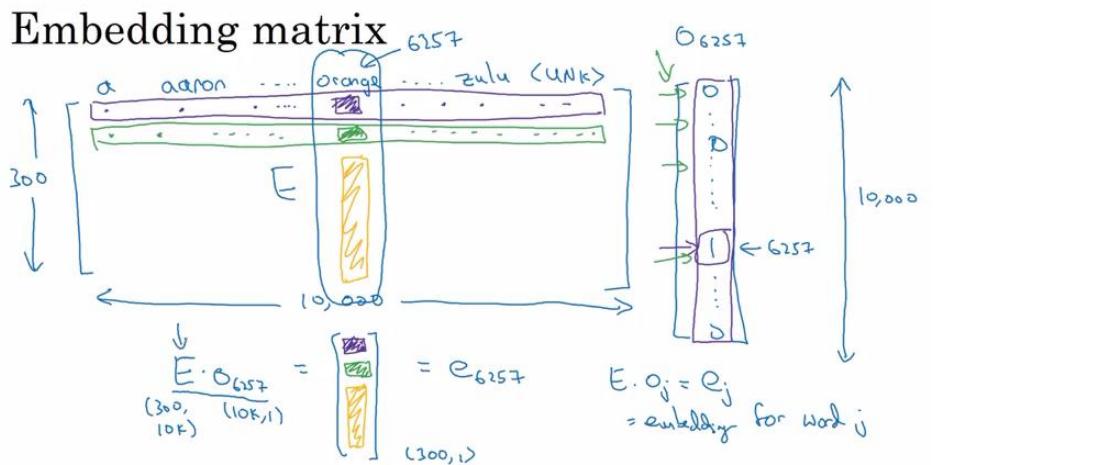
$e_{boy} - e_{brother} \approx e_{girl} - e_{sister}$

$e_{boy} - e_{brother} \approx e_{sister} - e_{girl}$

embedding-matrix (其实就是 embedding 组成的 matrix)

learn embedding matrix E

embedding-matrix 乘以 该 word 的 onehot 得到 embedding vector e



E 就是将每个 word 的 embedding e 按列排列！！ $(300, 10000)$

$E * \text{Onehot}(10000, 1)$ 得到 e $(300, 1)$

其实得到的向量就是 E 的对应 word 的列，因为 onehot 其他元素为 0，只有一个 1，对应 E 的那一列

in practice 不会这样计算，onehot 太长了，矩阵计算效率太低

我们只需定位到 E 的正确索引列就好！！！

keras 有 embedding layer，可以直接得到 E matrix

You have trained word embeddings using a text dataset of m_1 words. You are considering using these word embeddings for a language task, for which you have a separate labeled dataset of m_2 words. Keeping in mind that using word embeddings is a form of transfer learning, under which of these circumstance would you expect the word embeddings to be helpful?

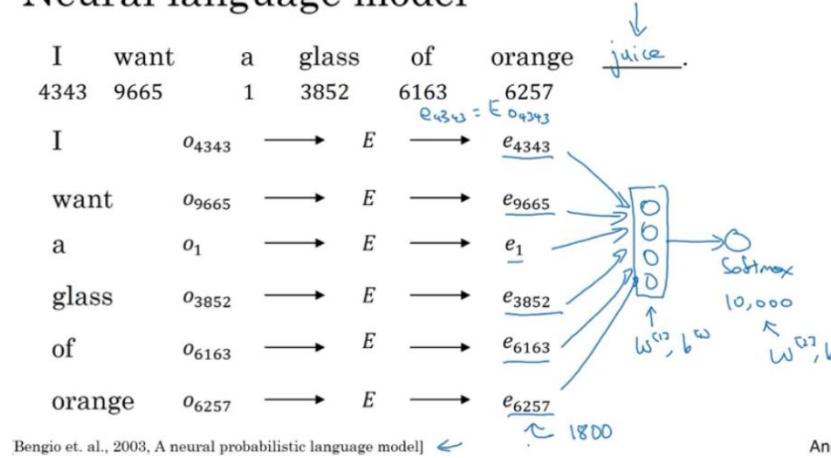
$m_1 \gg m_2$

2.2 learning-word-embeddings

Neural language model

[Bengio et. al., 2003, A neural probabilistic language model]

Neural language model



Bengio et. al., 2003, A neural probabilistic language model

Andrew Ng

我们想推断 juice target: 想预测的 word

1 先把每个 word onehot

2 乘以同一个 embedding matix 得到每个 word 的 embedding e
(每个 word uses same E matrix! ! !)

3 然后将一句话的 embedding 给 network output 预测下个 word onehot。节点个数是 len(vocabulary)

E 矩阵也作为一个参数可学习变量, 还有每个 embedding 和 fully-connect 的参数, 不断后向传播更新

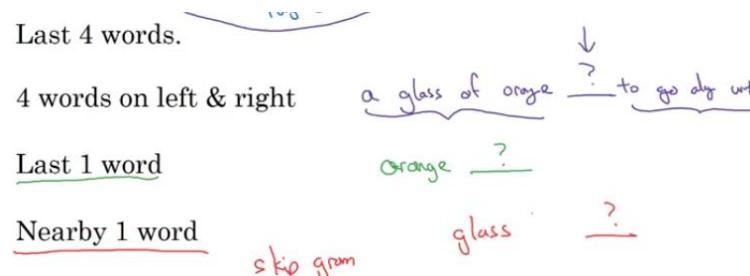
context 是前面的 word, 必须 fixed historical window=6 , 前 6 个 word, 来预测第 7 个 word

最终得到 embedding! ! ! !

other target/context pair choose for learning embedding

不同的 target/context pair 的选择, context 和 target 的定义

context embedding 作为 input 给神经网络, target 是预测的 word 来学习 E Matix



可以前 4 个预测第五个
可以前后预测中间

context: target 周围的 word

很多类型的 context:

- 1 last n word 前面的几个 words
- 2 4 word left and right
- 3 前一个 word
- 4 隔壁一个 word (skip gram)

如果为了 language model 只需要 last n word

如果为了 embedding，可以用上面任意方法

word2vec (learn embedding 更简单有效)

[Mikolov et. al., 2013. Efficient estimation of word representations in vector space.]

embedding 是 supervised learning 问题，需要 context 是 feature, target word 是 y

Skip-grams model (nearby word)

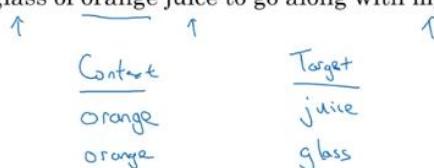
先得到 context-target pair

1 第一步 random pick context word 2 然后在一定窗口区间 (+-5) 内，随机选取一个 target word(第重复几次)

会产生很多可能的 pair:

Skip-grams

I want a glass of orange juice to go along with my cereal.



这些 pair 我们认为都是正确的 context-target pair (near by 都是 context)
因为，用 supervised learning 来预测 target，并不在乎准确预测(准确的一个 target word)，二是为了学习 word 直接的距离(经常出现在一起)

When learning word embeddings, we create an artificial task of estimating

$P(\text{target} \mid \text{context})$

. It is okay if we do poorly on this artificial prediction task; the more important by-product of this task is that we learn a useful set of word embeddings.

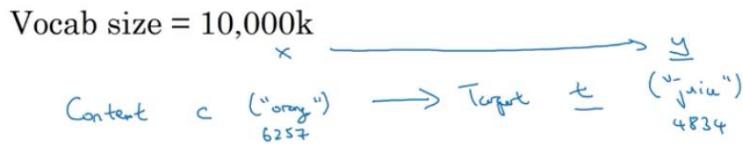
(Skip-grams model 缺点：有些 word 本身 frequency 太高，不能只 sample 这些 (下一节))

In the word2vec algorithm, you estimate $P(t | c)$, where t is the target word and c is a context word. How are t and c chosen from the training set? Pick the best answer.

- c is the sequence of all the words in the sentence before t .
- c is the one word that comes immediately before t .
- c and t are chosen to be nearby words.
- c is a sequence of several words immediately before t .

Model detail

Model



中间是 neron network，我们想从 context word 输出 target word

$$o_c \rightarrow E \rightarrow e_c \rightarrow o \xrightarrow{\text{softmax}} \hat{y}$$

$e_c = E o_c$

结构：

content word onehot 乘以 E matrix (参数) 得到 embedding，再到 fully-connect

softmax 输出值向量是 $p(t|c)$ ：在给定 c 下，每个 word(节点)是 t 的概率

$$\text{Softmax: } p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

θ_t = Parameter associated with output t

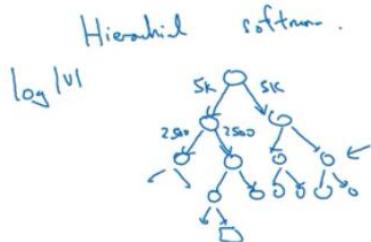
θ_t 是每个 target (t) 节点的连接前一层 embedding 层的参数向量

embedding 直接到 fully-connection

但 softmax 计算量太大 to sum，每个 example 都要计算最后每个节点都要 sum 一遍，太慢了，而且 len(vocabulary) 一般几十万

hierarchical 分层 softmax:

question? ? 没讲清楚
word 的序号是前 5000 后 5000 前 2500 后 2500 直到每个真实值



设计 tree 时: common word 需要放上面, 不常见 word 放 deep 处
解释:

softmax 是 n 个 classifier

而 hierarchical softmax

其实每一个节点也是一个 classifier, 用来判断 target 的位置(5000 前? 后),
将 classifier 以树的形式组织, 最终叶节点得到 target word

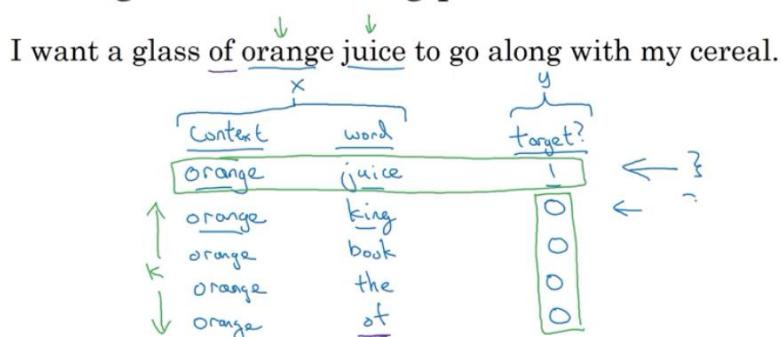
Negative-sampling 更有效的算法

[Mikolov et. al., 2013. Distributed representation of words and phrases and their compositionality]

定义一个新的 supercised learning problem: 新的 X,y
将 softmax 转换成多个 logistic classification 二分类问题

given a pair words (context, target), predict 是否是 context-target pair (0 或 1)

Defining a new learning problem



1 sample **positive** example:

跟上面一样 skim-gram 一样

1 random pick a context word

2 random **pick target word** with in window of context word (+-5word)

附近的 word。只要再窗口范围内就是 target

2 sample k **negative example**:

1 random pick context word

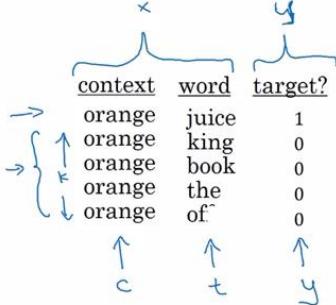
2 pick a random word in dict (理论上是负的，也有可能错是 target)
dataset 大时 k= 2-5 小时 5-20

2 训练; X 是 word pair y 是 target

Model

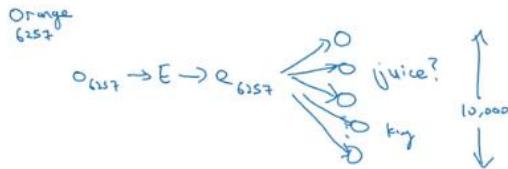
$$\text{Softmax: } p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

$$P(y=1 | c, t) = \sigma(\theta_t^T e_c) \leftarrow$$



$\theta^T e_c$ 是最后一层 fully-connect 的输出值, 可以用来计算 cross entropy: $p(t|c)$, 也可以用来计算 sigmoid $p(y=1|c, t)$, 其实是一样的 $p(t|c)$ 越大, 我们认为 $p(y=1|c, t)$ 越大

现在的算法是 model $p(y=1|c, t)$, 用 sigmoid 激活



将 softmax 变成独立的 1000k 个 binary logistic regression, lr 模型特别好 train!
因为每个 lr model 参数除了 E, 就只有 len(embedding) 个 fully-connect 参数了

每个节点值是出现 word pair(c, t) 而 $y=1$ 的概率

每个 iteration, 用这 $k=1$ 个 example train logistic model

我们只需要 train k 个 binary logistics regression, 容易训练!!!

不再需要再计算 softmax, 计算量更小!!!!

并且, 我们同时只 train 上面 $k+1$ 个 example

然后再生成 $k+1$, 再训练....

解决随机选取频率高的 word 问题

注意: sample negative example:

不能根据频率来, the a of 出现太多了

$f(w)$ 是 word frequency of w

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10,000} f(w_j)^{3/4}}$$

按照 $P(w)$ 来 sample word

glove-word-vectors (另一种方法)

另一种方法 learn embedding ! ! !

[Pennington et. al., 2014. GloVe: Global vectors for word representation]

(global vectors for word representation)

我们之前对 context word 定义: immediate word before the target word

I want a glass of orange juice to go along with my cereal.

orange 是 context word juice 是 target word

此时定义 context 和 target: 两个 word 是否 close (+-10word 内)

count frequency: X_{ij}

c, t

$X_{ij} = \# \text{times } i \text{ appears in context of } j.$

$$X_{ij} = X_{ji}$$

衡量两个 words close to each other, X 越大, 越相关!

Log X 也是单调递增, 衡量 2 个 word 相关性, 取 log 更加正态

X 相当于是我们的 target

Model detail:

$$\text{minimize} \quad \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} \left(\Theta_i^T e_j - \log X_{ij} \right)^2$$

$\Theta_i^T e_j$
" $\Theta_t^T e_c$ "

此时的 learning objective:

可以通过我们得到的最后一层得到输出： $\theta^T e_j$ ，用来逼近近似 Log X 值，也就

$\theta^T e_j$ 是输出两个 word 的相关性值 Log X

X 相当于是我们的 target，theta 和 e 是学习的参数

让 theta * e 来学习，i 和 j 的相似度！！！

cost 目标 让差值最小即可

$$\text{minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j - \log X_{ij})^2$$

"weight term" $f(X_{ij})$
 $f(X_{ij}) = 0 \text{ if } X_{ij} = 0, \text{ "0 log 0" = 0}$

修正：

如果 $X_{ij}=0$, $-\log X_{ij}$ 就是负无穷，loss 没办法计算

加个 weight term $f(X_{ij})$ 条件函数：

1 当 $X_{ij}=0$, $f(X_{ij})$ 也是 0, $-\log X_{ij} = 0$, 就不会有影响！只 sum, 至少出现过一次的 X_{ij}

2 $f(X_{ij})$ 需要我们选择：对于 frequency 小的 word 多给 weight, 对于高频词汇：the of that 给小的 weight

$$\text{minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j + b_i + b'_j - \log X_{ij})^2$$

$$\text{minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j + b_i - b'_j - \log X_{ij})^2$$

这个算法的一大特色是要求 theta 和 e 是 symmetric 向量，

e of word final 是均值！！！

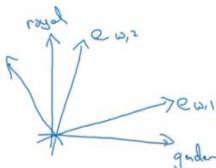
$$e_w^{(\text{final})} = \frac{e_w + \bar{e}_w}{2}$$

question? ? ? ? symmetric

2.3 featurerization view of embedding

A note on the featurization view of word embeddings

	Man	Woman	King	Queen
(5391)	(9853)	(4914)	(7157)	
Gender	-1	1	-0.95	0.97
Royal	0.01	0.02	0.93	0.95
Age	0.03	0.02	0.70	0.69
Food	0.09	0.01	0.02	0.01



$$\text{minimize } \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j + b_i - b'_j - \log X_{ij})^2$$

$$(A\theta_i)^T (A^T e_j) = \theta_i^T A^T A e_j$$

Andrew Ng

其实是找到了 word 的内在 feature
embedding feature 不一定都能解释
可能会混合很多种可解释的 feature

2.4 embedding 是需先单独 train from large text

2.5 applications-using-word-embeddings

1 sentiment classification

monitor the comment:

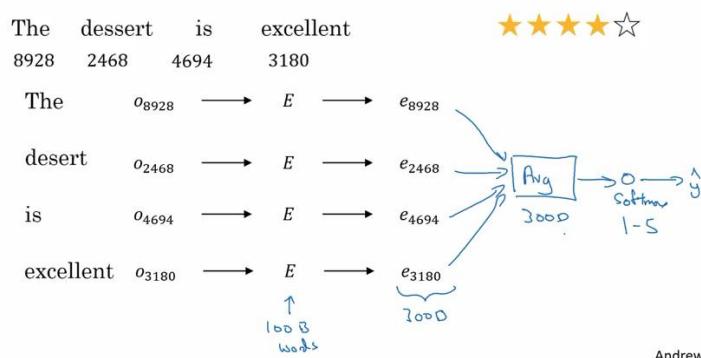
Sentiment classification problem

x	y
The dessert is excellent.	★★★★★☆
Service was quite slow.	★★☆☆☆☆
Good for a quick meal, but nothing special.	★★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆☆

最大的 challenge : not enough label dataset

但有了 embedding, 及时小的 dataset (少量 label data), 效果也好！！

Simple sentiment classification model



用 onehot 乘以我们已经学习到的 E matrix, 得到 embedding

Average or sum e vector(任意句子长度) ---然后连接 full-connect softmax

对应 5 个 star rate

只是 sum 或 average 每个 word 的 meaning, 效果不是很好

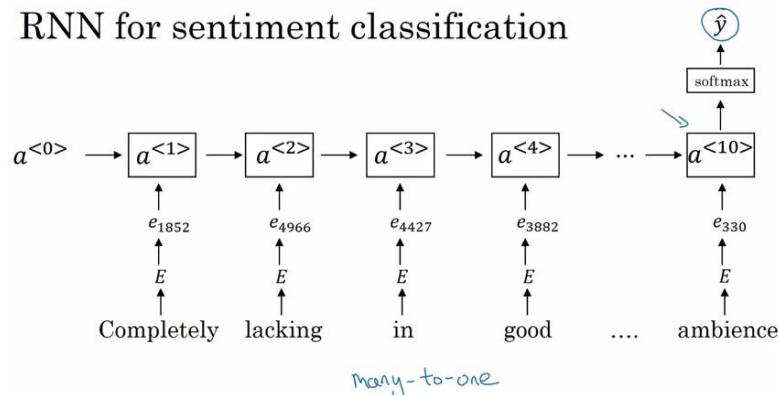
忽略了 word order 和上下文环境

“Completely lacking in good taste, ~~good~~ service, and ~~good~~ ambience.”

不使用

用 RNN 来解决:

RNN for sentiment classification



RNN 记住 word 的顺序和上下文环境意思

我们选一些少量的 train sentence, 进行训练, 就像下面, 每个 word

先乘以 E, 得到 embedding, 然后 feed to RNN, 最后一个 time step predict!!

训练好的模型, 对应有些没有在 train vocabulary 的 word, 同样效果好, 因为已经有了 embedding, 有了相似性!

Suppose you download a pre-trained word embedding which has been trained on a huge corpus of text. You then use this word embedding to train an RNN for a language task of recognizing if someone is happy from a short snippet of text, using a small training set.

x (input text)	y (happy?)
I'm feeling wonderful today!	1
I'm bummed my cat is ill.	0
Really enjoying this!	1

Then even if the word "ecstatic" does not appear in your small training set, your RNN might reasonably be expected to recognize "I'm ecstatic" as deserving a label $y = 1$.

2 debiasing-word-embeddings

diminish the bias in word embedding 不是机器学习的 bias, 而是真正的偏见和歧视

[Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings]

embedding 的局限:

我们要用 ML model 来做重要的 decision, 因此, 不能有 bias!!!
embedding 可以学习到 bias, write by people!! 人的偏见, 性别 bias, 职业 bias, 因为会有某些偏激的作者

The problem of bias in word embeddings

Man:Woman as King:Queen

Man:Computer_Programmer as Woman:Homemaker X

Father:Doctor as Mother:Nurse X

Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model.

会有一些不好的结果, 会学到 bias!!!

我们希望 man 和 woman 都是 programmer, 而不区分(bias)

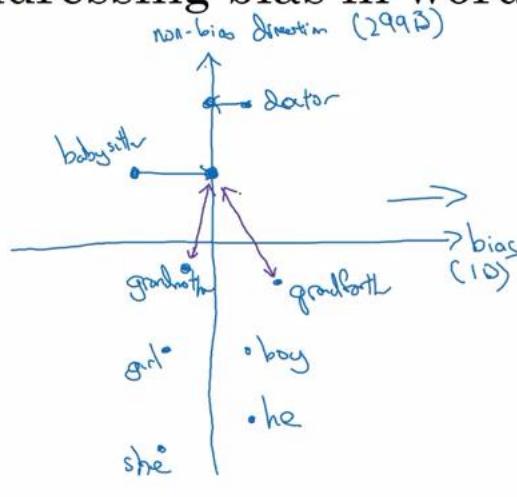
消除:

1 identify bias direction(例如 gender) question? ? ?

先找到 direction

找几个 word embedding 相减(特点是, 两个 e 都只是相差性别), 并 average
需要用 svd 分解来解决!!

Addressing bias in word embeddings



1. Identify bias direction.

$$\begin{cases} \mathbf{e}_{he} - \mathbf{e}_{she} \\ \mathbf{e}_{male} - \mathbf{e}_{female} \end{cases} \rightarrow \text{average}$$

2. Neutralize: For every word that is not definitional, project to get rid of bias.

3. Equalize pairs.

grandmother grandfather
girl boy

例如: 横坐标代表性别 bias 方向

有些 word 如 granpa granma 本身就具备性别属性, 有 bias 上的差异正常!

2 中立化

有些不具备性别如 doctor, baby, 需要 neutralize 中立化 无 gender 偏见
doctor, baby 需要在 non-bias direction 上投影, 得到新的 embedding, 就没有 bias 方向的差异

想

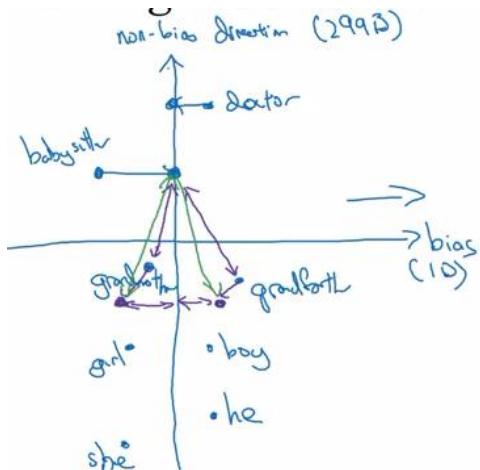
3 equalize pair

可能需要手工挑选出 word pair

granpa granma or boy girl 唯一不同是 gender

保证这一对 word 和其他投影的 word (baby, doctor) 的 dist 一样！！

会调整 move pair word 使得 1 到 non-bias axis 轴的距离一致，2 并且到 babysitter 和 doctor 距离一样



train a classifier identify which word is definitional (male female), which word are non-definition

3 sequence-models-attention-mechanism

3.1 various-sequence-to-sequence-architectures

basic-models

machine translate

[Sutskever et al., 2014. Sequence to sequence learning with neural networks] ↩

[Cho et al., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation]

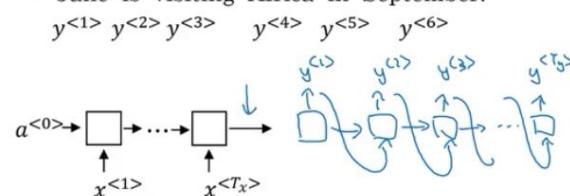
翻译：法-英

X sequence y 也是 sequence

Sequence to sequence model

$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad x^{<4>} \quad x^{<5>}$
Jane visite l'Afrique en septembre

→ Jane is visiting Africa in September.



encoder network: 左边黑色 fit X sequence

decoder: 右边蓝色 output y

train set: sentence pair

image caption

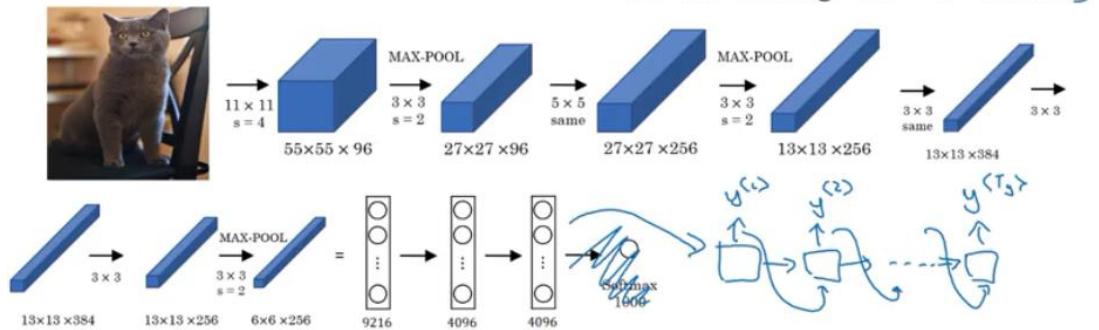
[Mao et al., 2014. Deep captioning with multimodal recurrent neural networks]

[Vinyals et al., 2014. Show and tell: Neural image caption generator]

[Karpathy and Fei Fei, 2015. Deep visual-semantic alignments for generating image descriptions]

input image output sentence

Image captioning



先卷积，得到最终的 **fully layer** 特征向量 encoding，当做 encode，然后给 RNN decode，生成 caption

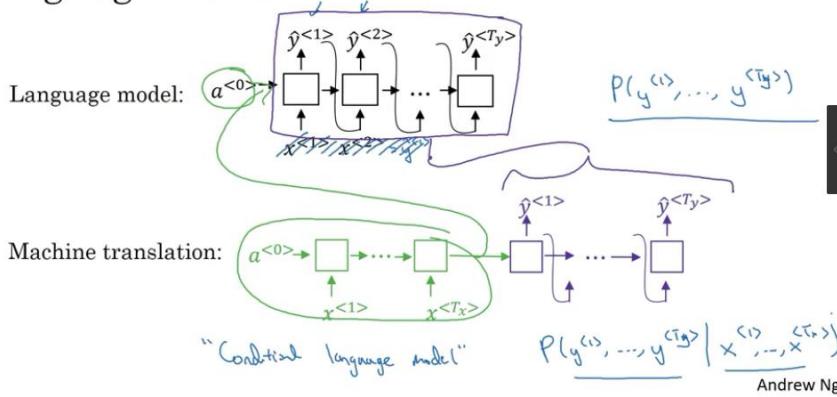
3.2 picking-the-most-likely-sentence

conditional language model

我们不想 random choose translation 或者 caption，而是 best! !

LM: 给出一个句子的概率，初始化状态是 0 向量

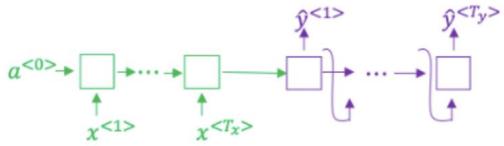
Machine translation as building a conditional language model



而机器翻译：已经有了法语的 encode，作为初始状态值，后面的 decoder 和 LM 一样

又也称为 **条件 LM**，给定法语为条件，一句话的概率！！

Consider using this encoder-decoder model for machine translation.



This model is a “conditional language model” in the sense that the encoder portion (shown in green) is modeling the probability of the input sentence x .

- True
- False

我们要找到概率最大的条件概率！！！而不是 random sample 一句话

Finding the most likely translation

$$\text{Jane visite l'Afrique en septembre.} \quad P(y^{<1>} , \dots, y^{<Ty>} | x)$$

- Jane is visiting Africa in September.
- Jane is going to be visiting Africa in September.
- In September, Jane will visit Africa.
- Her African friend welcomed Jane in September.

$$\arg \max_{y^{<1>} , \dots, y^{<Ty>}} P(y^{<1>} , \dots, y^{<Ty>} | x)$$

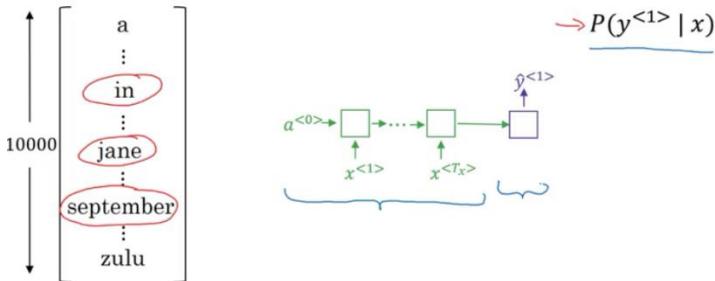
greedy search: 每次增加一个 word 找到条件概率最大的, 需要遍历每个 word!!! 不合适！！！

beam-search

beam width 同时考虑 3 个概率最大的 words top3

Beam search algorithm $B = 3$ (beam width)

Step 1



先看下一个 word 的条件概率 (第一个 softmax 输出) 然后 keep top 3 words (概率最大的三个)作为 y_1 的 candidate

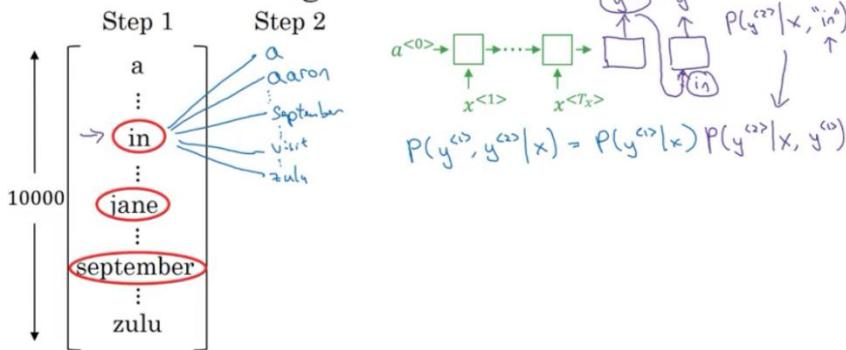
然后只需要考虑这 3 个 word 的分别下个最大概率的 word: $3 * 10000$, 而不是考虑全部 word: $10000 * 10000$, 选 $p(y_1, y_2 | x)$ 最高的三个 word 作为 y_2 备选!

这个过程有可能 y_1 的备选会被舍弃

有可能三个是 (in with) (in a) (jane are) september 舍弃

如何计算条件概率？

Beam search algorithm



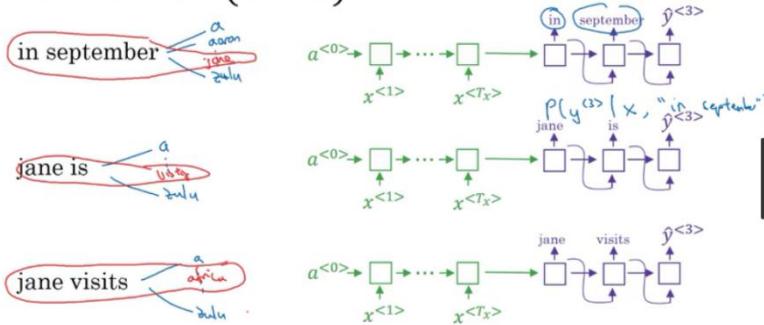
RNN 对于 y_2 输出概率是 $p(y_2 | x, y_1)$

乘以前面的 $p(y_1 | x)$, 得到 $p(y_1, y_2 | x)$ chain rule

$p(y_1 = "in" | x)$ 和 $p(y_2 = \text{1000 个 word} | x, y_1)$ 都已知, 计算 $p(y_1, y_2 | x)$, 并选出 top 3

得到第三个输出 $p(y_3 | x, y_1, y_2)$ 再乘以 $p(y_1, y_2 | x)$ 得到 $p(y_1, y_2, y_3 | x)$

Beam search ($B = 3$)



$$P(y^{<1>}, y^{<2>} | x)$$

对于每个 step 的三个 word 只需要 copy 上面网络结构 3 个每一 step 全部计算完之后, 选出 top3

refinements-to-beam-search

改进：条件概率最大，条件概率是 chain rule 乘积

Length normalization

$$\arg \max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

$$\log \arg \max_y \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

1 因为 p 概率都是小于 1, 相乘得到很小的数, 数字稳定性, 太小, 而不能正确存储表示, 因此我们取 log 的和, log 是严格增函数

2 句子越长, 相乘的概率也小, 取 log 后因为 p 小于 1, $\log p$ 小于 0, 句子

越长，越负，也不合理，这样只会输出短的翻译
需要 normalize 下，除以句子长度！！！

$$\frac{1}{T_y} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

或者 soft normal, between full normal 和不 normal, 要 try alpha 值

$$\frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

$\alpha = 0.7$ $\alpha = 1$
 $\alpha = 0$

how to choose B

B, 每个 step 的备选词数

B 越大 发现更好的翻译的概率越大，但计算量越大
多试一下 b 的值

Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for $\arg \max_y P(y|x)$.

error-analysis-in-beam-search

对于翻译不是很好，进行调整

两个组成部分：1RNN 2 Beam serach 到底哪一部分有问题
增大 Beam num 不会 hurt performance，但很费时间

Example

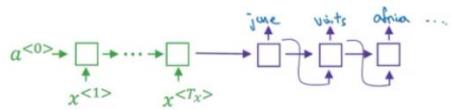
\rightarrow RNN
 \rightarrow Beam Search

Jane visite l'Afrique en septembre.

Human: Jane visits Africa in September. (y^*)

Algorithm: Jane visited Africa last September. (\hat{y}) \leftarrow

RNN computes $P(y^*|x) \leq P(\hat{y}|x)$



上面 human 是理想翻译 y star，算法是机器翻译 y hat

用 RNN model 计算 $p(y \text{ hat} | x)$ $p(y^* | x)$ 来比较大小(带入模型和 x 来计算概率)

分两种情况：

Error analysis on beam search

Human: Jane visits Africa in September. (y^*) $P(y^*|x)$

Algorithm: Jane visited Africa last September. (\hat{y}) $P(\hat{y}|x)$

Case 1: $P(y^*|x) > P(\hat{y}|x)$ $\arg \max_{\hat{y}} P(\hat{y}|x)$

Beam search chose \hat{y} . But y^* attains higher $P(y|x)$.

Conclusion: Beam search is at fault.

如果 $P(y^*|x)$ 更大说明 RNN 有效

本应该选 y^* , 却选了 \hat{y} , 说明 beam 有问题

Case 2: $P(y^*|x) \leq P(\hat{y}|x) \leftarrow$

y^* is a better translation than \hat{y} . But RNN predicted $P(y^*|x) < P(\hat{y}|x)$.

Conclusion: RNN model is at fault.

Andrew Ng

本应该 y^* 大于 \hat{y}

RNN 有问题

Error analysis process

Human	Algorithm	$P(y^* x)$	$P(\hat{y} x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	2×10^{-10}	1×10^{-10}	B
...	...	—	—	R
...	...	—	—	Q
				R
				R
				:

Figures out what fraction of errors are “due to” beam search vs. RNN model

Andrew!

多选几个翻译不好的 bad example, 来统计, 看哪个错误更多

如果是 RNN 增加 regularization 或更换网络结构

如果是 B, 就调整 b 的大小

bleu-score-optimal

[Papineni et. al., 2002. A method for automatic evaluation of machine translation]

bleu score 用来给 machine translation 进行评分

bleu : bilingual evaluate understudy 替代

Evaluating machine translation

French: Le chat est sur le tapis.

→ Reference 1: The cat is on the mat. ←
→ Reference 2: There is a cat on the mat. ←
→ MT output: the the the the the the the the
Precision: $\frac{7}{7}$ Modified precision: $\frac{2}{7} \leftarrow \begin{matrix} \text{Count}_{\text{clip}}(\text{"the"}) \\ \text{Count}(\text{"the"}) \end{matrix}$

Bleu
bilingual evaluation methodology

reference 是人理想翻译

MT output: the the

precision: 看翻译结果的 word 是否在我们的 reference 里，
分母是 output 某个 word 的个数，分子是 output 的每个 word 是否出现在 2 个
reference 里会得到 7/7 不合理，bad 翻译

modeify:

the 在 ref1 出现 2 次，在 ref2 出现 1 次

分母是翻译某个 word 的出现次数，翻译的每个 word 取出现次数最高 max(clip)
的 reference 次数作为分子

以上只是得到单个 word 的 score

我们关注多个 word 的 score

两个 word 的词组

Bigram(两个词) 两两一组为单位

Bleu score on bigrams

Example: Reference 1: The cat is on the mat. ←

Reference 2: There is a cat on the mat. ←

MT output: The cat the cat on the mat. ←

	Count	Count _{clip}	
the cat	2 ←	1 ←	
cat the	1 ←	○	
cat on	1 ←	1 ←	
on the	1 ←	1 ←	
the mat	1 ←	1 ←	

1 先计算 output 出现频率 count

2 clip count: reference 那个出现频率高，取哪一个！

求和再求比值 clip count/output count

formalize

Unigram(单个单词)

$$P_1 = \frac{\sum_{\substack{\text{unigrams} \in y \\ \text{Unigram}}} \text{Count}_{clip}(\text{unigram})}{\sum_{\substack{\text{unigrams} \in y}} \text{Count}(\text{unigram})}$$

n-gram(n个单词)

$$P_n = \frac{\sum_{\substack{n\text{-grams} \in y \\ \text{n-gram}}} \text{Count}_{clip}(n\text{-gram})}{\sum_{\substack{n\text{-grams} \in y}} \text{Count}(n\text{-gram})}$$

都是衡量 output 和 reference 的相似度！！如果结果好， p_1, p_2, p_3, p_n 都为 1

Bleu:

计算多个 p_n : $p_1, p_2, p_3 \dots$, 然后 combine 一起计算综合 bleu score

还要乘以一个 bp factor

bp(brevity penalty), 因为对于 output 越简短, p_n 值会越高。我们要 normalize 下用 bp

bp 函数定义:

Bleu details

p_n = Bleu score on n-grams only

Combined Bleu score: $\text{BP} \exp\left(\frac{1}{4} \sum_{n=1}^4 P_n\right)$

P_1, P_2, P_3, P_4

$\text{BP} = \text{brevity penalty}$

$$\text{BP} = \begin{cases} 1 & \text{if } \text{MT_output_length} > \text{reference_output_length} \\ \exp(1 - \text{MT_output_length}/\text{reference_output_length}) & \text{otherwise} \end{cases}$$

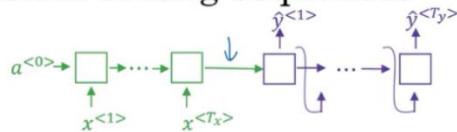
也用于 image caption！！

3.2 attention-model-intuition

1 人类翻译，不是一下子记住全部句子再翻译，二是一部分一部分的翻译

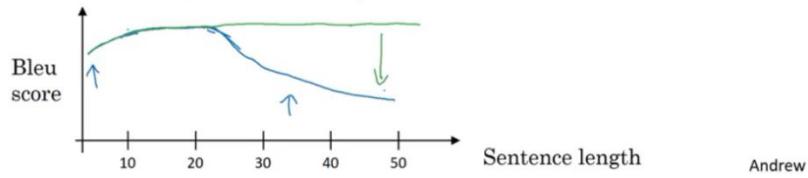
对于 long sentence bleu score 很低，因为跟人工翻译的结果不同，人工是一部分一部分翻译！！！机器是全部句子读进去再翻译

The problem of long sequences



Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.



Andrew

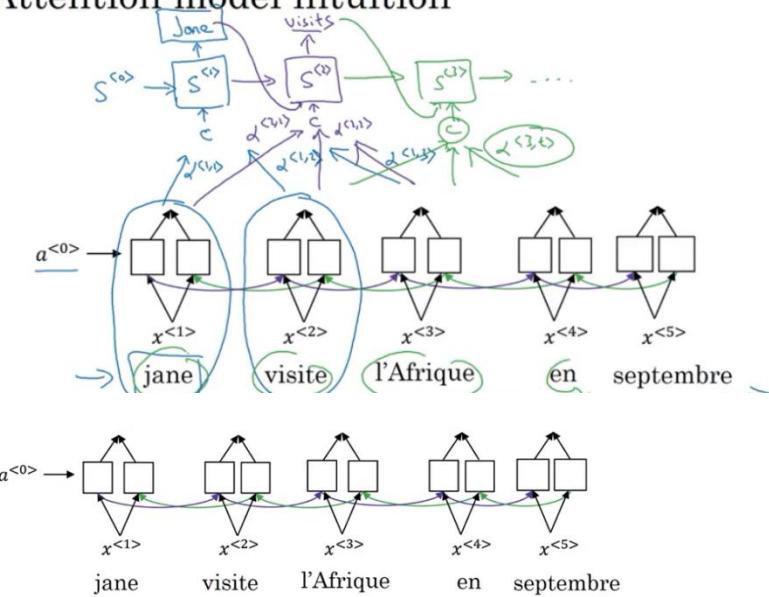
2 记住这么长的 sequence 不是容易 for RNN!! BLUE SCORE 会下降!

attention model

[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

[Xu et. al., 2015. Show attention and tell: neural image caption generation with visual attention]

Attention model intuition

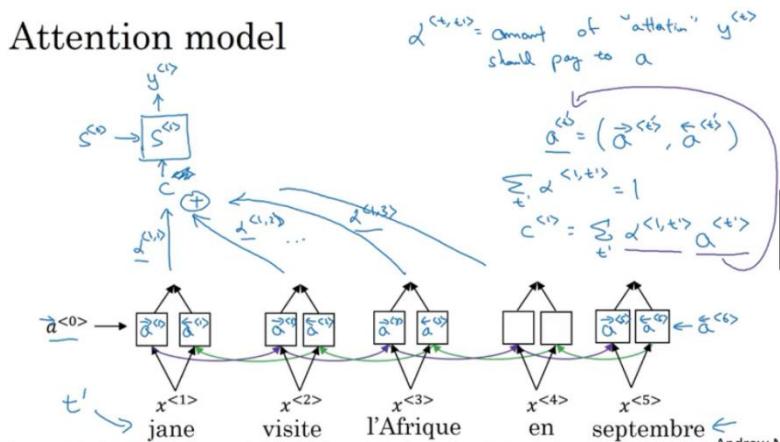


input 是双向 bi-RNN 用来计算 feature

output 翻译也是单向 RNN

其实就是两个 RNN 或 (LSTM) 的叠加，两层 LSTM, 一层接受 input, 一层 output, 输出层通过 **attention weight** 接受 input 层的 activation

Attention model



attention weight: how much attention should pay to the **part** of input words to **generate the translated word**

某些 input 乘以 attention weight alpha 得到 c context, 再给 output 层会有一个 local window pay attention to 多少 words, 就跟人翻译一样！！！
 t' 代表 input 的 t 时刻, t 是第几个 output y

alpha< t, t' > 是第 t 个翻译 $y^{(t)}$, 需要对第 t' 时刻的 $a^{(t')}$ pay 的 attention(weight)

attention weight alpha 约束: 一个翻译 word y 的 attention wight alpha 合为 1

$$\sum_t \alpha^{(t,t')} = 1$$

第一层的 a 其实是两个向量合并一个向量, a forward 向量, a backward 向量

$$\underline{\alpha}^{(t)} = (\overrightarrow{\alpha}^{(t)}, \overleftarrow{\alpha}^{(t)})$$

c 的计算:

$$c^{(t)} = \sum_{t'} \alpha^{(t,t')} \frac{a^{(t')}}{\alpha}$$

$$\underline{\alpha}^{(t)} = (\overrightarrow{\alpha}^{(t)}, \overleftarrow{\alpha}^{(t)})$$

$$\sum_t \alpha^{(t,t')} = 1$$

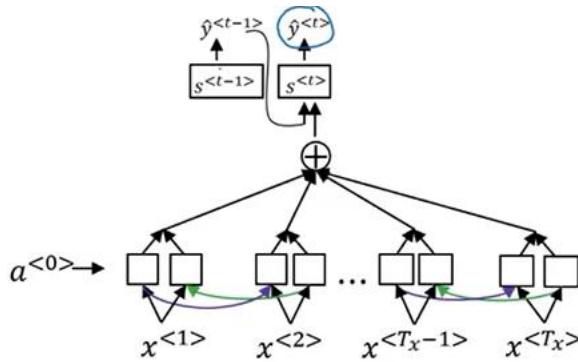
$$c^{(t)} = \sum_{t'} \alpha^{(t,t')} \frac{a^{(t')}}{\alpha}$$

alpha< t, t' > 其实就是一个数值, alpha* $a^{(t')}$ broadcast 到向量每个元素。所有 alpha* $a^{(t')}$ 再累加得到 c (context), 得到维度等同于 a 的 context！！

权重确定

alpha< t, t' > depend on $s^{(t-1)}$ 和 $a^{(t')}$

$s^{(t-1)}$ 和 $a^{(t')}$ 决定了给当前 $a^{(t')}$ 对 state $s^{(t)}$ 多少权重！！！我们不知道函数关系



就 train a model, 不是直接学 alph, 而是学 e(energies) 其实是 z, 然后通过 softmax 变化得到 alpha! ! !

通过 \$s^{<t-1>}\$, \$a^{<t>}\$ 得到 \$e^{<t, t'>}\$, 然后 softmax 得到 a

s 向量和 a forward 向量, a backward 向量, 3 向量拼成一个 vector! ! !

$$\rightarrow \underline{\alpha^{<t,t'>}} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

其实就是多加一个 dense layer

\$\alpha^{<t,t'>}\$ 是向量由 \$\alpha^{<t,1>}\$、\$\alpha^{<t,2>}\$、\$\alpha^{<t,3>}\$... 元素值组成

train a function from \$s^{<t-1>}\$, \$a^{<t>}\$ 到 \$e^{<t, t'>}\$

得到:

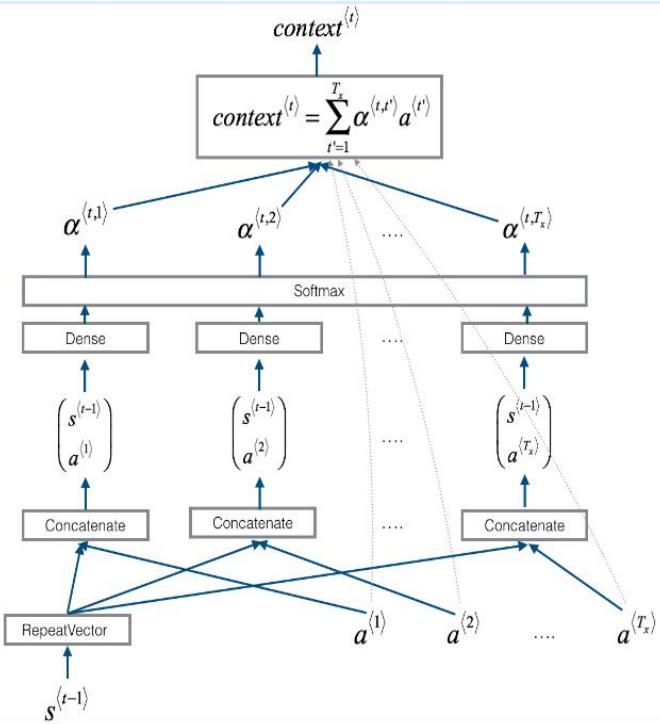
但运行时间太长, quadratic time

3.3 attention 和传统 LSTM 不同(不依赖前一 step 的输出)

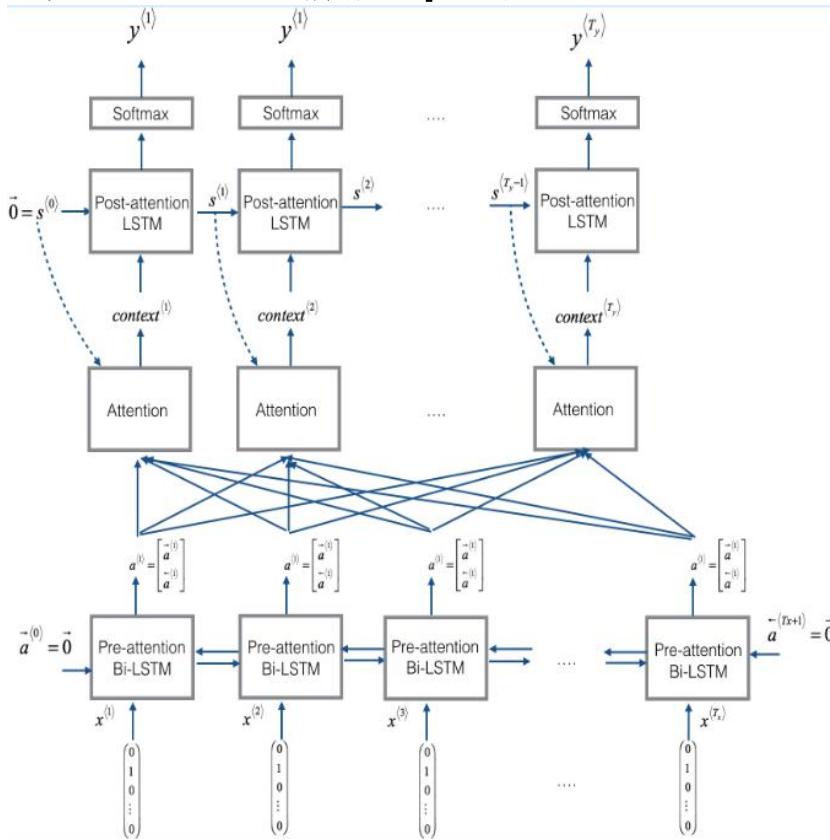
post-activation LSTM at time \$t\$ does not take the specific generated \$y^{<t-1>}\$ \$y^{<t-1>}\$ as input; it only takes \$s^{<t>}\$ \$s^{<t>}\$ and \$c^{<t>}\$ \$c^{<t>}\$ as input

因为 here isn't as strong a dependency between the previous character and the next character in a YYYY-MM-DD date. 输出并不依赖前 1 step 的输出, 因此跟以往的 LSTM 不同!

context 的详细计算



直接加了 dense 层，所有 step 公用一个 dense 层



应用在其他方面：

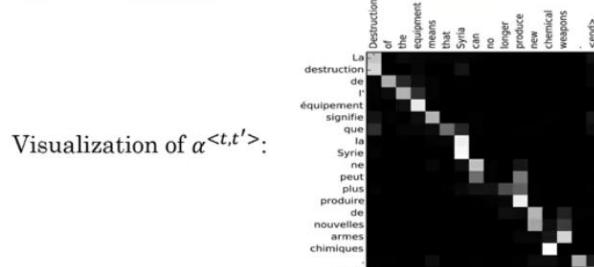
1 image caption

[Xu et. al., 2015. Show, attend and tell: Neural image caption generation with visual attention]

2 data normalization:

July 20th 1969 → 1969 - 07 - 20

23 April, 1564 → 1564 - 04 - 23



越白，代表 generate word 时更需要哪个 input word, attention! ! !

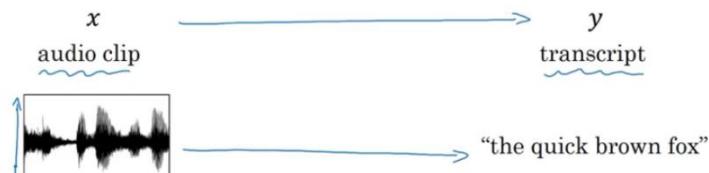
3.3 speech-recognition-audio-data

speech-recognition(CTC)

phonemes: hand-engeering audio feature extract

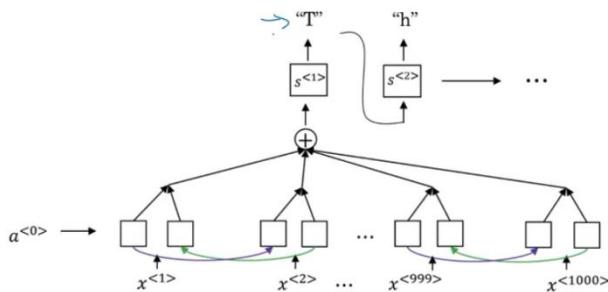
对于 deep learning 没有太大意义, 不需要 hand engeer

Speech recognition problem



10000h-1000000h audio to train

Attention model for speech recognition

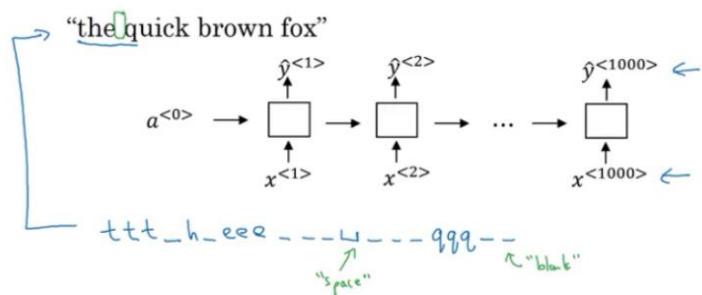


CTC Cost ? ? ? question 没听明白 如何提取 speech clip 特征?
如何给 RNN

[Graves et al., 2006. Connectionist Temporal Classification: Labeling unsegmented sequence data with recurrent neural networks]

CTC cost for speech recognition

(Connectionist temporal classification)



模型：input output sequence length 一样长！

实际模型是用 bi-direction RNN with more hidden layer

这里只是简化

而且 input timestep 会很长几十分钟

input 特别长，但 output 只有几个单词，如何处理：

CTC 允许生成 output：

让 output 也变大，，但最终的有效 transcript 小

允许重复字母，blank

Basic rule: collapse repeated characters not separated by "blank"

消除重复的不被 blank 隔开的字符

得到 the q

Under the CTC model, identical repeated characters not separated by the "blank" character (_) are collapsed. Under the CTC model, what does the following string collapse to?

_c_o_o_o_kk__b_o_o_o_o_o_o_kkk

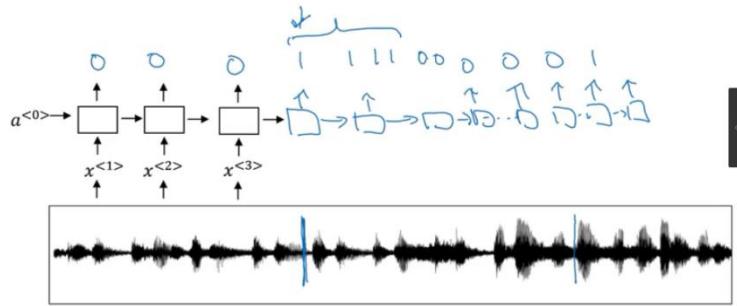
- cokbok
- cookbook
- cook book
- coookboooooooookkk

02_trigger-word-detection

唤醒设备 trigger-word

imbalance

Trigger word detection algorithm



第一个竖线，是 trigger word 刚结束，开始标 label 为 1，代表需要执行在这一点之前的全部标记 0

下一次还是这样

可以只标记一次 1，也可以一直标记 1，具体要看效果

In trigger word detection, $x^{<t>}$ is:

- Features of the audio (such as spectrogram features) at time t .
- The t -th input word, represented as either a one-hot vector or a word embedding.
- Whether the trigger word is being said at time t .
- Whether someone has just finished saying the trigger word at time t .