

上两周收到反馈



小伙伴纷纷反映

所以这讲以治愈为主，回血！

从零开始的深度学习课程

III 小试牛刀

谢晋璟

Commerce Data Science Team

CDL数据科学社区（原BDA社区）
数据科学工作室
技术茶会

2016-07-29

目录

- 1 文字空间的魔术：词向量嵌入
- 2 钢铁之心：DeepDream
- 3 人人都是艺术家：Neural Style

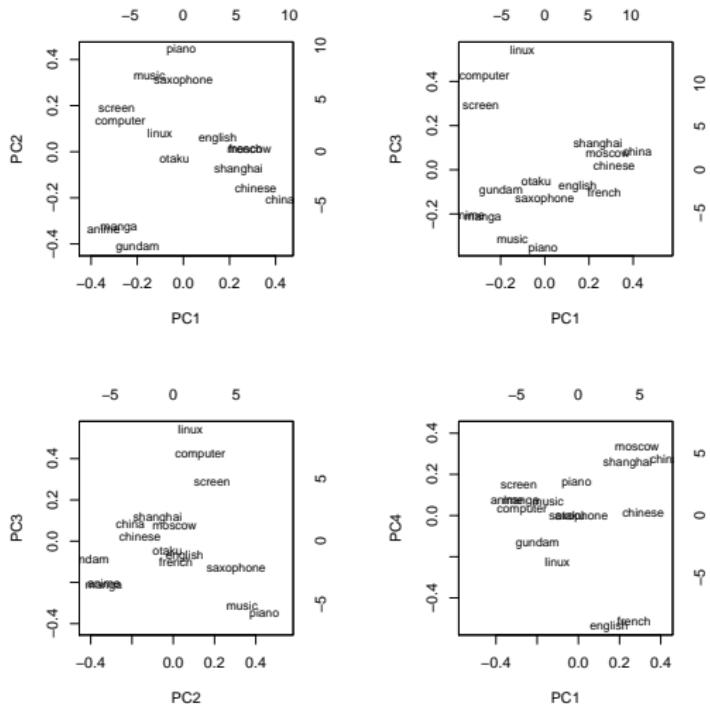
目录

1 文字空间的魔术：词向量嵌入

2 钢铁之心：DeepDream

3 人人都是艺术家：Neural Style

Word Embedding



单词上的算术

类比的问题

$$\text{king} - \text{man} + \text{woman} = ?$$

类比的问题：答案

$$\text{king} - \text{man} + \text{woman} = \text{queen}$$

词的联想

如果词和词之间可以做算术，那么我们从一个词想到另一个词，可以看作是找一个和这个词距离比较近的词。

词的联想：例子

看到Einstein，你会想到什么？

有趣的例子：从“Einstein”联想到的

Word	Cosine distance
Albert_Einstein	0.733897
relativity_theory	0.585443
Richard_Feynman	0.579787
Poincaré	0.579148
Niels_Bohr	0.575201
Satyendra_Nath_Bose	0.574414
Roger_Penrose	0.573074
Godel	0.572241
Feynman	0.569784
Erwin_Schrodinger	0.568730
Kurt_Godel	0.565212
Neils_Bohr	0.556486
Stephen_Hawkins	0.554454
E_=_mc2	0.554421
pinto_stallion_named	0.553572
quantum_theory	0.552305
Michanowski	0.551879
e_=_mc2	0.547430
Leibniz	0.545091
wife_Mileva_Maric	0.544715
Edwin_Hubble	0.542130
Einstein_theories	0.541579
Erwin_Schrodinger	0.540633
Einstein_Theory	0.533239
Schrodinger	0.527533
Fritz_Zwicky	0.526305
physicist_Albert_Einstein	0.526228
Paul_Dirac	0.524826
Leo_Szilard	0.524282

Italian Noodle(意大利面条)

Word	Cosine distance
Sicilian	0.568733
Ravioli	0.561519
Pasta	0.555687
Grilled_Ham	0.551191
Tuscan	0.549648
Neopolitan	0.549221
Spuntino	0.548061
L'_Italiano	0.543655
banh_mi_sandwiches	0.542087
Mortadella	0.539787
Pizza	0.538253
Napoletana	0.537879
Spinach_Pie	0.535580
Frank_Falcinelli	0.535369
Celestino_Drago	0.534901
Sole_Mio	0.533439
Consomme	0.531142
molto_bene	0.528877
Undici	0.527219
La_Tavola	0.526726
Vero_Amore	0.524769
Roasted_Duck	0.524135
Italian_Sausage	0.522686
Stir_Fry	0.521774
Spasso	0.521771
Neapolitan	0.521657
Grilled_Eggplant	0.521130
Trattoria_Roma	0.519969
steaks_chops_seafood	0.519557

Chinese wasabi(中国芥末)

Word	Cosine distance
Japanese	0.620969
Sichuan_peppercorns	0.616808
steamed_dumplings	0.610281
gyoza	0.610182
char_siu	0.608616
jiaozi	0.601028
China	0.599032
kung_pao_chicken	0.595360
gai_lan	0.587701
hairy_crab	0.585646
soy_sauce	0.583847
oyster_sauce	0.583586
ma_po_tofu	0.583204
Szechuan_cuisine	0.583097
pan_fried_dumplings	0.581520
Hunanese	0.580145
sticky_rice	0.580119
shoyu	0.578296
fermented_tofu	0.578209
kimchi_spicy	0.577994
Sichuanese	0.577959
stir_fried_beef	0.577672
stinky_tofu	0.576045
grilled_eel	0.575835
hoisin_sauce	0.575758
wheat_flour_noodles	0.572134
stir_fried_dishes	0.571494
umeboshi	0.571301
wasabi_paste	0.571265

otaku-宅

Word	Cosine distance
anime	0.688199
Otaku	0.686353
manga	0.668271
cosplay	0.641386
anime_manga	0.632793
otakus	0.627822
doujinshi	0.615456
yaoi	0.607665
manga_ka	0.602957
otaku_culture	0.596741
gaijin	0.596525
Dragon_Quest	0.591386
mangaka	0.590400
shojo_manga	0.588483
manga_comics	0.581097
cosplayers	0.577647
Akihabara	0.576508
kawaii	0.566935
Densha_Otoko	0.565963
manga_anime	0.564245
salaryman	0.560964
shojo	0.560443
Reiji	0.560213
Otobe	0.558439
Kabukicho	0.558185
kawaii_cute	0.558029
Tsubasa	0.557664
Ikebukuro	0.557141
yokai	0.557067

Gundam–高达

Word	Cosine distance
Mobile_Suit_Gundam	0.702465
Neon_Genesis_Evangelion	0.640670
Panzer_Dragoon	0.634481
Macross	0.633355
Gundam_anime	0.625981
Samurai_Champloo	0.621379
anime	0.620331
Evangelion	0.617715
mecha	0.616069
Dragon_Quest	0.615190
Sengoku_Basara	0.609764
Dragonball_Z	0.599007
Super_Robot_Taisen	0.598813
dungeon_crawler	0.596829
Xenosaga	0.594272
Fullmetal_Alchemist	0.592895
Casshern	0.592079
Macross_Plus	0.591188
Bubblegum_Crisis	0.589902
Tetsujin	0.589206
PSOne	0.588089
Naruto_RPG	0.587908
Gunbuster	0.587292
Tatsunoko	0.587285
Metal_Gear	0.586781
Suikoden	0.585780
Mega_Man	0.584908
Onimusha	0.584542
Crystal_Chronicles	0.582533

tensor—张量

Word	Cosine distance
uniaxial	0.652096
τ	0.641997
θ	0.633397
ϕ	0.624644
wavefunction	0.624107
dimensionless	0.623830
perturbation_theory	0.618628
enthalpy	0.617326
Young_modulus	0.613836
shear_modulus	0.613501
σ	0.612916
quadratic	0.611249
eigenvalues	0.609803
Fig._1a	0.608887
anisotropy	0.607120
dipole_moment	0.605125
anisotropic	0.602005
ellipsoid	0.601188
electron_diffraction	0.600229
adiabatic	0.598648
magnetisation	0.598623
calorimetric	0.598458
π	0.596860
Nernst	0.596301
Fig._2b	0.595193
ν	0.594749
antiferromagnetic	0.590304
tetragonal	0.590037
η	0.589546

obama–奥巴马

Word	Cosine distance
mccain	0.731902
hillary	0.728460
obamas	0.722964
george_bush	0.720568
barack_obama	0.704584
palin	0.704312
clinton	0.693445
clintons	0.681683
sarah_palin	0.681515
john_mccain	0.680071
barack	0.672264
gop	0.669116
biden	0.668955
dick_cheney	0.664339
dems	0.664307
repubs	0.659749
BHO	0.656987
hillary_clinton	0.654193
reid	0.653470
reagan	0.650150
americans	0.648017
ron_paul	0.647283
cheney	0.643558
pelosi	0.636424
washington	0.632059
america	0.631153
libs	0.629876
whitehouse	0.626063
washington_dc	0.625762

类比

上面的结果来源于word2vec 网站上训练好的模型
 《GoogleNews-vectors-negative300.bin.gz》

原理

输出在词库中距离最近的词

再来看看词类比的例子: (king - man) + woman

Word	Distance
queen	0.711820
monarch	0.618968
princess	0.590243
crown_prince	0.549946
prince	0.537732
kings	0.523684
Queen_Consort	0.523595
queens	0.518114
sultan	0.509859
monarchy	0.508741
royal_palace	0.508717
throne	0.500580
royal	0.493820
Princess_Sikhanyiso	0.493662
ruler	0.490928
empress	0.488781
Prince_Paras	0.483294
princes	0.481082

(man - girl) + love = ?

Word	Distance
passion	0.517466
loved	0.492245
despise	0.488416
loves	0.484867
hate	0.484220
adore	0.479200
admire	0.471034
detest	0.442191
loving	0.430838
kindness_gentleness	0.428300
undying_loyalty	0.425971
unbridled_passion	0.424251
dislike	0.421701
passionate	0.421190
gentleness_kindness	0.416584
undying_love	0.415577
loathe	0.414302
hates	0.413525
cherish	0.407323
I'ma_diehard	0.405887
revere	0.405413
undying_passion	0.404867
abiding_affection	0.404383
God	0.403339
devotion	0.402409
Yea_yea	0.402100
fanship	0.401773
many Splendored	0.399846
paraphrase_Oscar_Wilde	0.399173

(Shanghai - China) + Japan

Word	Distance
Tokyo	0.798192
Osaka	0.722244
Yokohama	0.718740
Nagoya	0.703933
Maebashi	0.649806
Japanese	0.629648
Fukuoka	0.624415
Shizuoka	0.621143
Ibaragi	0.620902
Toyko	0.616576
Nishinomiya_Hyogo_Prefecture	0.610960
Kishiwada	0.607163
Hachioji	0.604481
Takamatsu	0.603315
Osaka_Japan	0.601511
Aichi_Prefecture	0.599840
Kawagoe	0.596718
Ota_Ward	0.593317
Chiba_Prefecture	0.591857
Saitama	0.591763
Tokyo_Chiyoda_Ward	0.589471
Shizuoka_Prefecture	0.589426
Nishinomiya	0.589192
Gifu	0.588355
inTokyo	0.587857
Minato_Ward	0.587405
Eiji	0.585733
Kanagawa_Prefecture	0.585628
Tochigi_prefecture_north	0.583204

如何能让单词可以运算？

先要把它放到一个带有运算的集合上去—线性空间！

Naive的想法：one hot？

如果有 $50k$ 个单词，编号成 $1 \sim 50000$ ，那么每个单词不就对应到 \mathbb{R}^{50000} 里的一个向量了么？

问题是：这样的编码方式非常稀疏，词之间线性无关的，任何两个单词的**cosine similarity**都是0，这样就没法找最近的单词了。类比也一样，加减后**cosine similarity**非零的只有算式中的单词，没有任何类比性。

问题在哪？

嵌入的空间维数太高

如果它们能嵌入一个相对比较低维的空间，比如 \mathbb{R}^{100} ，那么每个词向量就不再是线性无关的了。然而如何让它们的关系反映实际语义上的关系？

这是一个降维问题？

找一个矩阵 W 将one hot的向量 x 映到低维的空间。

词和词的相似性

$$\text{sim}(x, y) = (y^T W^T) \cdot (Wx) = v_y^T v_x$$

如果一个句子当前位置的单词是 x_t , 它的上下文可以如下表示

$$[x_{t-2} \ x_{t-1} \ x_t \ x_{t+1} \ x_{t+2}]$$

我们希望 W 能使得上下文和当前位置单词关系尽量的大，和其他无关单词 y 关系尽量小

目标

$$\max_W \text{sim}\left(x_t, \frac{x_{t-2} + x_{t-1} + x_{t+1} + x_{t+2}}{4}\right)$$

and

$$\min_W \text{sim}\left(y, \frac{x_{t-2} + x_{t-1} + x_{t+1} + x_{t+2}}{4}\right), \forall y \neq x_t$$



实际的情况

上述假设嵌入是对称的：对于**context**的 x_I 和当前位置的单词 x_t 使用相同的 W 进行嵌入。这并不符合实际情况，所以我们实践中使用两个不同的嵌入 W' 和 W 。下面以**context**大小为4为例。

Context

$$\begin{aligned} v_I &= \frac{1}{4} W(x_{t-2} + x_{t-1} + x_{t+1} + x_{t+2}) \\ &= \frac{1}{4} (v_{t-2} + v_{t-1} + v_{t+1} + v_{t+2}) \end{aligned}$$

当前位置的词向量用 W' 做嵌入：

Current Word

$$v'_t = W' x_t$$

Loss Function

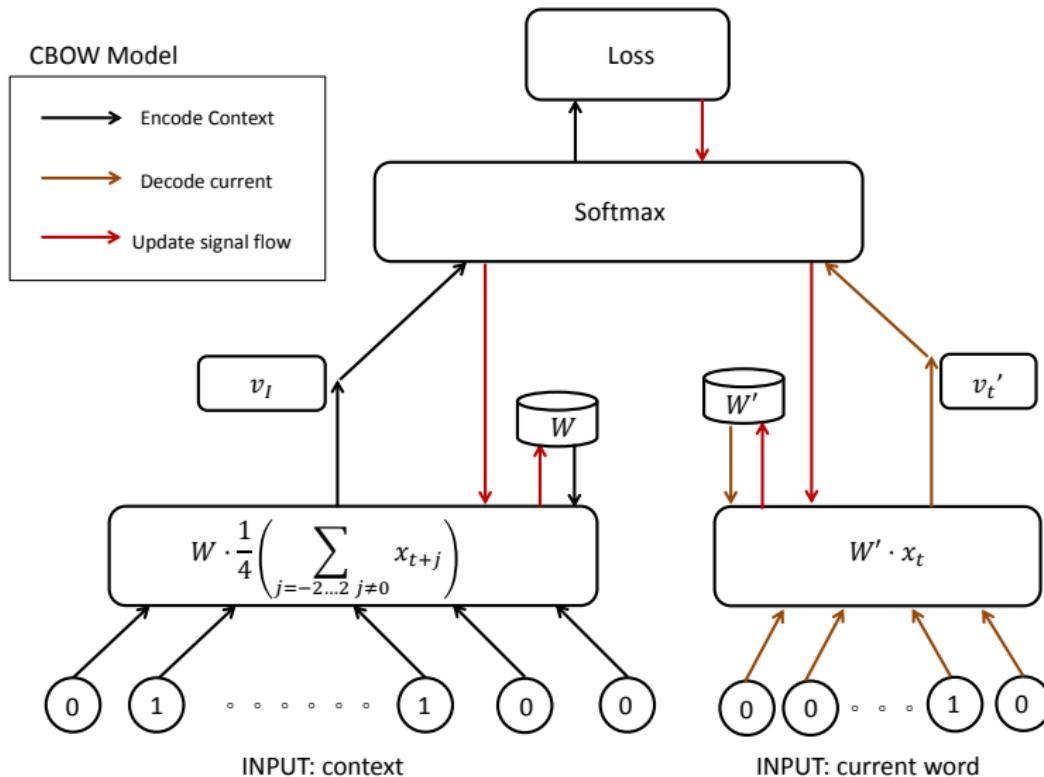
我们有很多句子作为训练数据，但是没有Label，所以这是无监督学习的问题，需要一个不依赖于标签的Loss Function。

Softmax Loss

$$\begin{aligned} L(x, W) &= -\log P(x_t | x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}) \\ &= -\log P(v'_t | v_{t-2}, v_{t-1}, v_{t+1}, v_{t+2}) \\ &= -\log \frac{e^{v'_t T v_I}}{\sum_y e^{v'_y T v_I}} \end{aligned}$$

这里 y 遍取Vocabulary里所有单词。

模型结构



两种常用模型

CBOW

上面的模型用**context**去预测当前位置的词，称为**CBOW(continuous bag of word)**模型

skip-gram

相反的，用当前的词去预测上下文的词，就是**skip-gram model**。相应的**Loss Function**是

$$-\sum_{-c \leq j \leq c, j \neq 0} \log P(v'_{t+j} | v_t)$$

这里的概率 P 仍然是个**softmax**

$$P(v'_j | v_t) = \frac{e^{v_j'^T v_t}}{\sum_y e^{v_y'^T v_t}}$$



实现的难点

巨大的softmax是困难所在

虽然这个模型本质是非常简单的单层神经网络，但是softmax的计算要对Vocabulary中每个词做计算，实际中并不现实。

有两种解决方法：

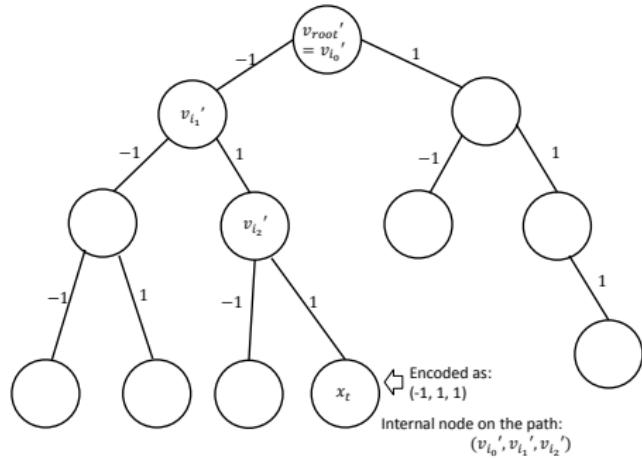
Hierarchical Softmax

用类似于Huffman码的方式把Vocabulary组织成二叉树。Vocabulary是叶子，每个internal的节点对应一个vector，得到observation的概率就是相应编码的路径给出的概率。

Negative Sampling

计算Softmax时不用所有的Vocabulary，而只是随机sample出一些，乘上相应的比例。

Hierarchical Softmax



如图：如果用 e_j 记录编码 $(-1, 1, \dots)$ ，那么 x_t 的概率就写成：

$$P(x_t) = \prod_{j=0}^{\text{depth}(x_t)-1} \phi \left(e_j \cdot v_{i_j}^{T'} v_I \right)$$

即解码部分不再用一个 W' ，而是用Huffman树代替了。

Negative Sampling

上面的方法修改了Decoding的部分，比较复杂，一个简单的方法就是每次就是softmax时，简单地取一个小样本作为负例就好了。用 v_O 记Observation对应的词向量， v_I 记Input的词向量。

Negative Sampling

$$L(v_O, v_I) = - \left(\log \phi(v_O'^T v_I) + \sum_{i=1}^k E_{y \sim P(y)} [\log \phi(-v_y'^T v_I)] \right)$$

这里 $P(y)$ 是频率分布的 $3/4$ 次幂：

$$P(y) \propto U(y)^{\frac{3}{4}}$$

演示网站 <https://ronxin.github.io/wevi/>

Config:

```
{"hidden_size":5,"random_state":1,"learning_rate":0.2}
```

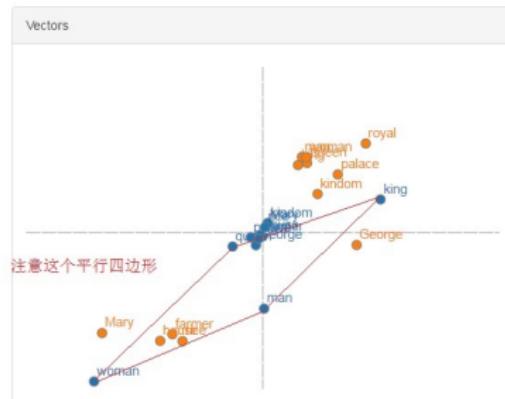
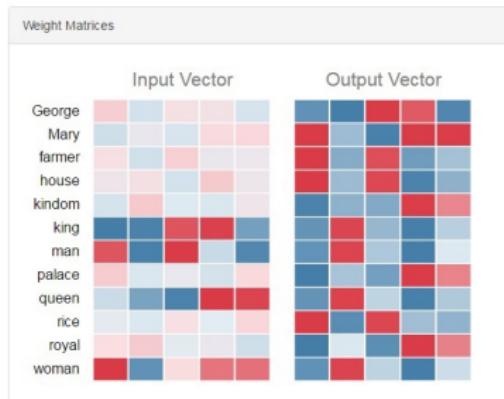
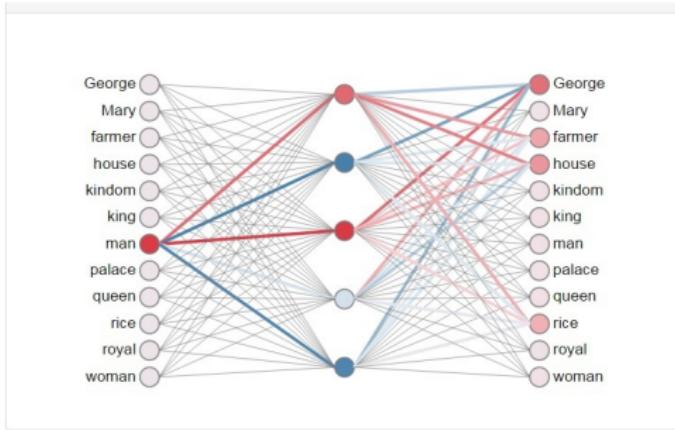
Training data (context|target):

```
king|kindom,queen|kindom,king|palace,queen|palace,king|royal,queen|royal,king|George,queen|Mary,man|rice,woman|rice,man|farmer,woman|farmer,man|house,woman|house,man|George,
```

Presets: King and queen

Update and Restart **Update Learning Rate**

Next 20 100 500 PCA



目录

1 文字空间的魔术：词向量嵌入

2 钢铁之心：DeepDream

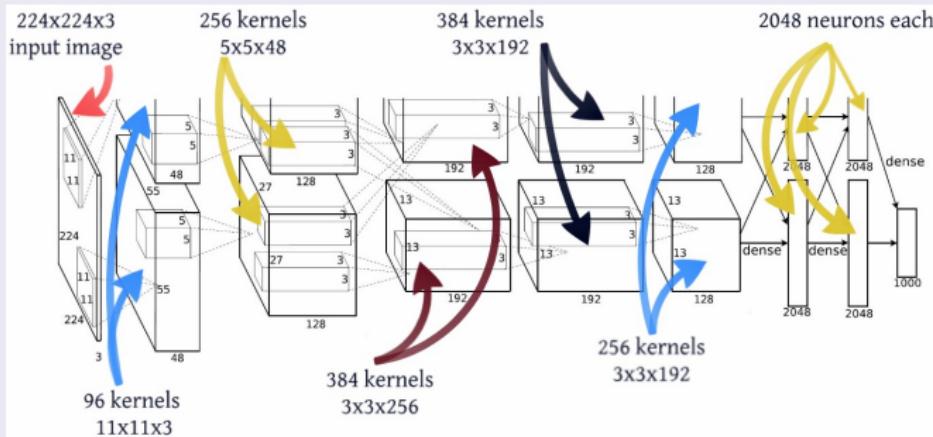
3 人人都是艺术家：Neural Style

谁的梦？

终于要开始看“Deep”的模型了

故事还是先从深度卷积网络讲起：

AlexNet



AlexNet可以说是从12年开始的这波深度学习热潮的最大的诱因之一。

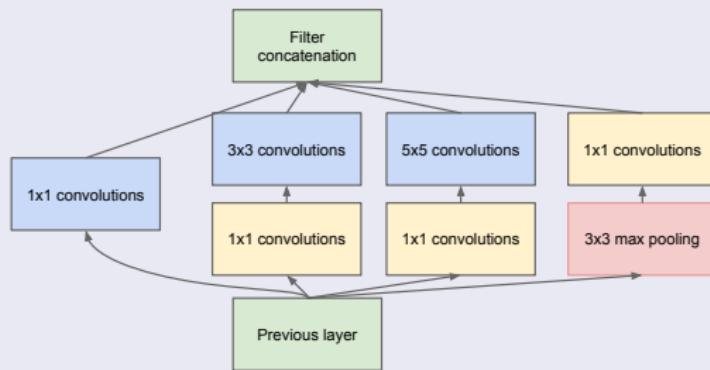


ImageNet Classification with Deep Convolutional Neural Networks

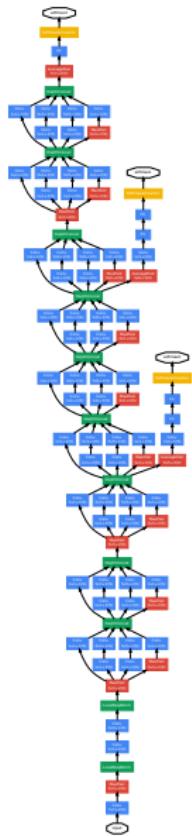
百花齐放的深度网络

在ILSVRC-2012的比赛上，AlexNet这个现在看来不太复杂的深度网络比当时的state-of-the-art要高出5%的准确率。传统特征工程做了十多年的工作被一口气突破，使得大量的CV的研究者纷纷转向DeepLearning。各种新的网络结构层出不穷。

GoogLeNet-Inception

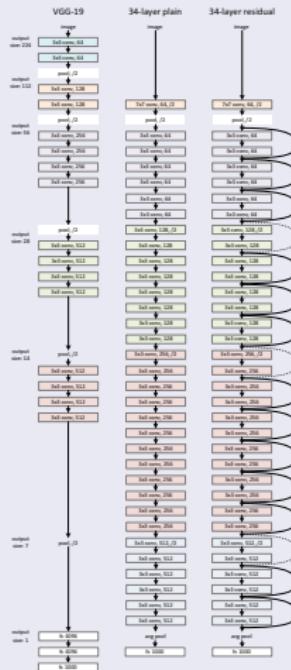


比如GoogLeNet用如上的Inception结构取代一个比较大的卷积变换，使得计算更快，用的参数更少。



更疯狂的有如大MS深达152层的ResNet (AlexNet只有8层!)

34层ResNet例子



DeepDream是可视化ConvNet的方法

DeepDream最早来自于Google Research Blog上的文章Inceptionism: Going Deeper into Neural Networks

它用一种直观的方法解释了
每层神经元的激活代表了什么？

卷积核的作用

学习好的卷积核的作用是提取图像的特征，不同的卷积核对应了不同的特征，所以每一层都会有很多的卷积核。

比如前面的AlexNet的第一层就有96个Kernel。

卷积核的输出

每个卷积层的输出代表当前输入区域对应相应的Kernel的激活度

解释：猫的Kernel

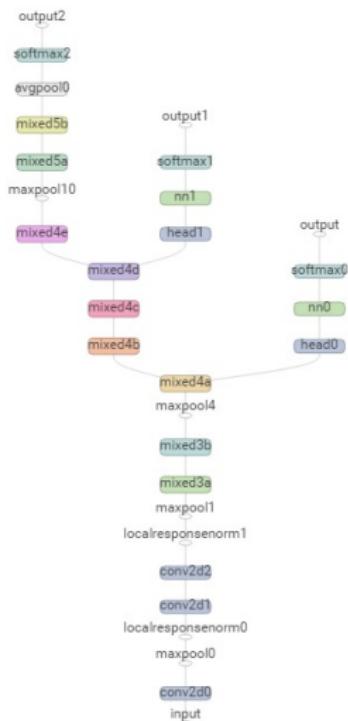
以“猫的Kernel”为例，如果输入区域有猫，那么经过这个Kernel后，相应的输出点的激活度就很大；反之输入区域如果很不像猫，那么输出点的数值就很小。

Feature和层数的关系

越低的层，代表的是越简单的特征—比如：线，面，边缘等。而越高的层，代表的是越复杂的特征—比如：上面提到的猫，狗等。

一些中间的层，则代表的是一些中间复杂的—比如：叶子，轮子等。

DeepDream的基本思想



通过BP算法反推出每个卷积核所提取的特征
前向的计算是从输入图像算出一个激活度。
我们反其道而行之，对某个中间层的神经加
一个很强的刺激，用BP算法计算这个反馈信
号对输入信号的影响

以GoogLeNet为例，mixed4d层139号特征



过滤高频分量得到的更自然的图

mixed4d_3x3_bottleneck_pre_relu, Feature number: 139



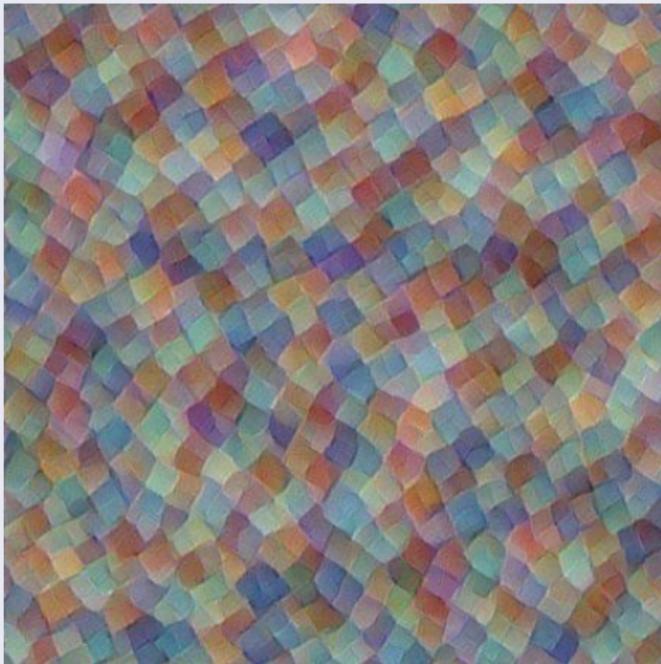
同一层上其他的Feature:

mixed4d_3x3_bottleneck_pre_relu, Feature number: 65



比较低的mixed3b层上的101号Feature:

mixed3b_1x1_pre_relu, Feature number: 101



混合139和65号特征：

mixed4d_3x3_bottleneck_pre_relu, Feature number: 65+139



把上面可视化的思想稍作变化，就得到DeepDream的算法

算法

输入 选定深度网络中的一层 L ，输入图像 I ，迭代次数 N

输出 处理后的图像 I

- 1 输入图像 I 向前传播到层 L
- 2 把这层上的激活当作是这层的反馈值 $\delta L = L$ （这相当于在这层做截断，加上 L_2 的Loss层）
- 3 用BP计算对输入的信号的反馈 $dI = BP(\delta L)$
- 4 更新输入图像

$$I \leftarrow I + \lambda dI$$

- 5 如果到达最大迭代次数 N 则结束，否则转2

实际中，还要对信号做一些扰动，以及使用一些技巧计算（不然对大图像会超过GPU的内存限制）。

原始图像



迭代30次



使用一个较浅的层



迭代390次—基本上已经“忘却”初始输入图像了



使用一个参考图像

如果反馈信号使
用如下参考图像
上的**Feature**来
激活



我们可以得到类
似右边的
的Dream

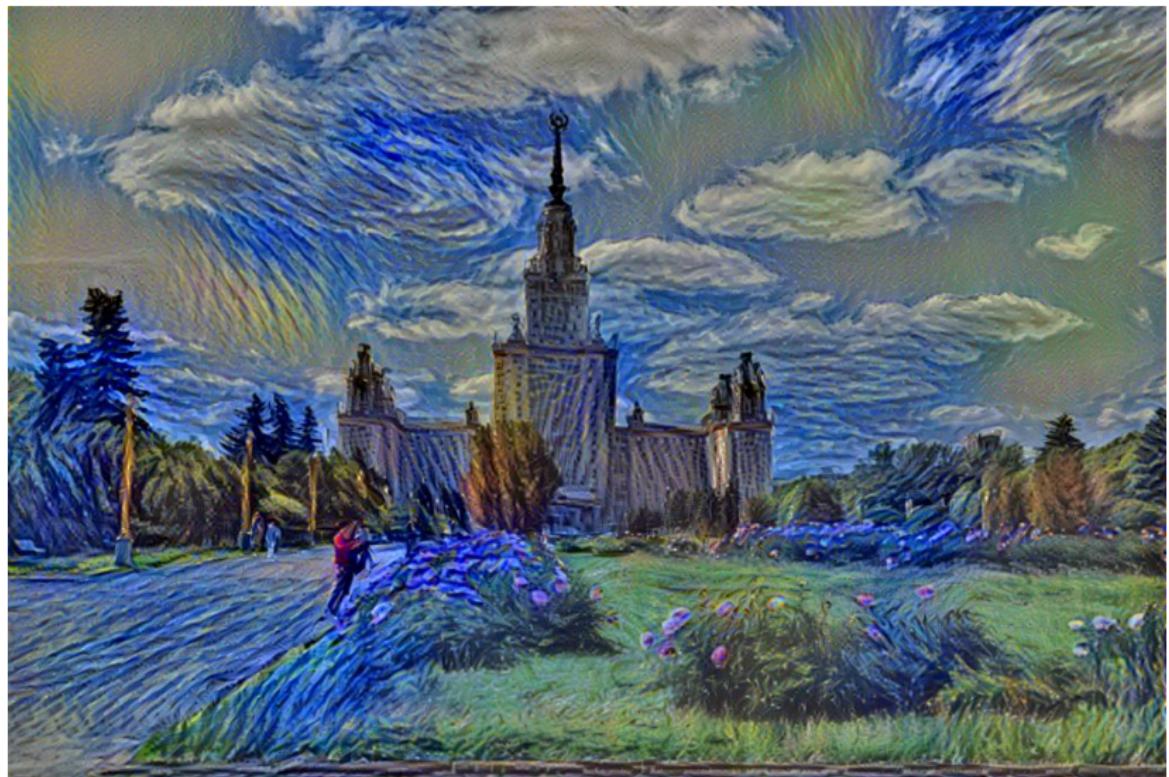


目录

1 文字空间的魔术：词向量嵌入

2 钢铁之心：DeepDream

3 人人都是艺术家：Neural Style



这可不是梵高隐藏的画作



好神奇？

左边是作为风格的名画，中间是照片，利用深度网络就得到右边的融合了最左油画风的照片。



+



=



梵高还是那个梵高，我们还可以玩点什么其他花样？

童心未泯

葱宝



+

星夜



=

葱宝的星夜



葱宝



+

小白楼



=

葱宝的小白楼



忧郁之兰

星夜



+

葱宝



=

星夜的葱宝



星夜



+

小白楼



=

星夜的小白楼



如何做到的？

核心

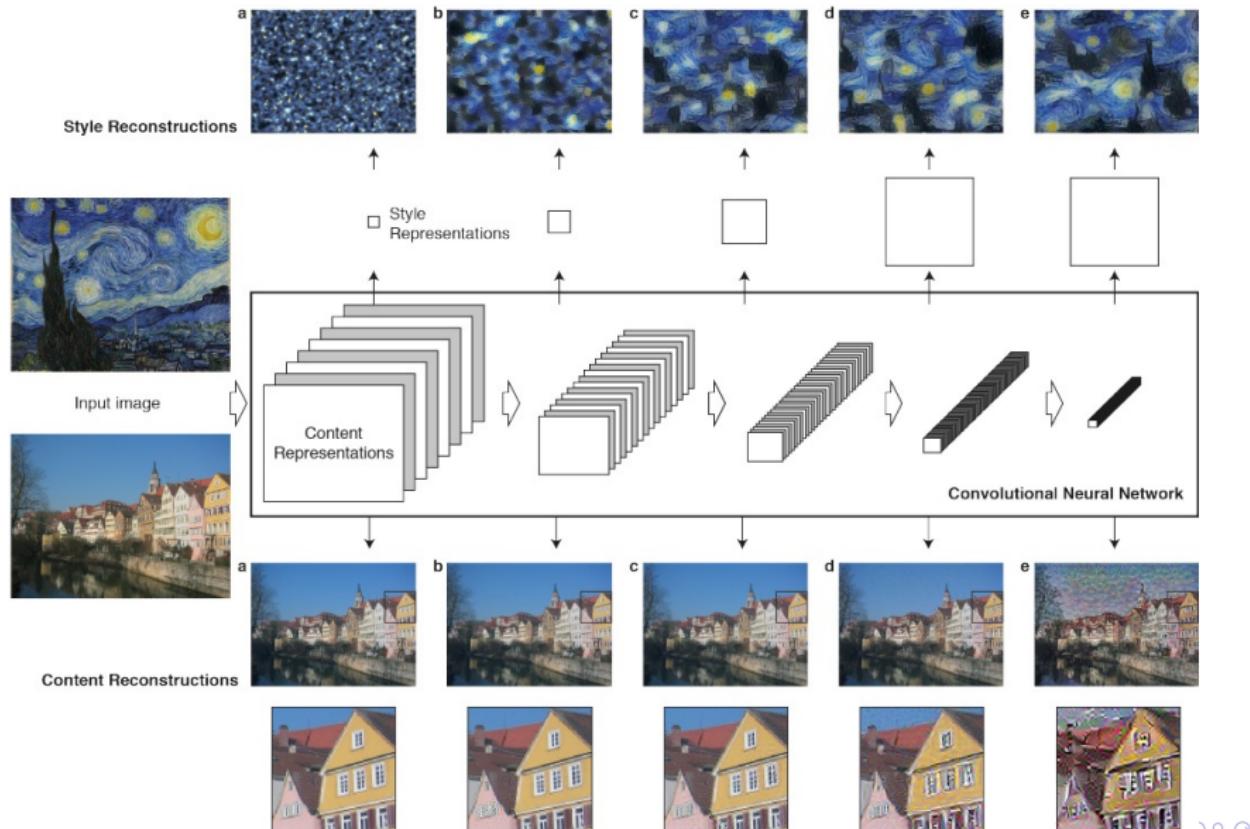
如何捕捉那飘渺不定的画风？

画风是什么

A Neural Algorithm of Artistic Style

15年的这篇文章开启了全新的思路，给了我们一个很有意思的答案。它使用了一个叫做**VGG-Net**的深度神经网络。**VGG-NET**原本是用于图像分类的，但是在这里却展示了“艺术细胞”？

原理图



数学一点地说

图像是二维平面上的向量场

用 I 记图像, p 记平面上的点, 那么图像是

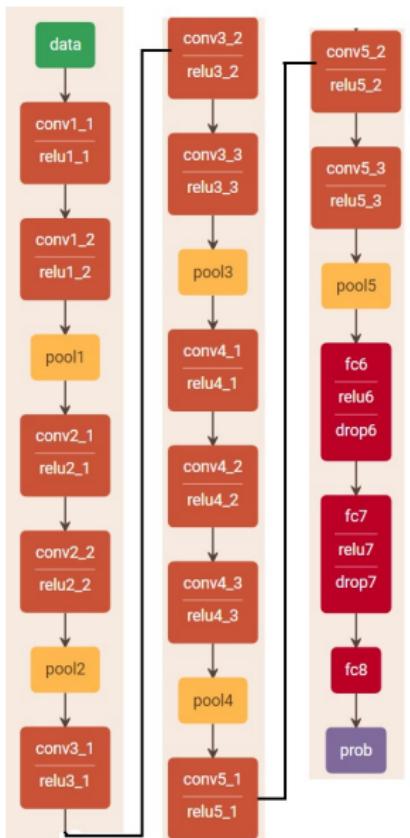
$$I(p) \in \mathbb{R}^3$$

回忆我们第一讲解释的卷积网络原理:

卷积核将一个向量场映射到另一个向量场

$$I^\ell = K^\ell \star I^{\ell-1}$$

中间层是非常高维的向量场, 比如VGG-NET。



卷积层 输出向量场所在的空间

conv1	\mathbb{R}^{64}
conv2	\mathbb{R}^{128}
conv3	\mathbb{R}^{256}
conv4	\mathbb{R}^{512}
conv5	\mathbb{R}^{512}

“画风”是中间层的场的特性

按照Neural Style文章的说法，选取conv1_1, conv2_1, conv3_1, conv4_1, conv5_1的层。计算Gram矩阵：

Gram矩阵

$$G_{fg} = I_f \cdot I_g$$

这里：

$$I_f = I_f(p), f \text{ is index of feature, } p \text{ is point on the plane}$$

一般地，Gram矩阵在数学上给出了从空间上的内积形式。而这里直观的意义是：它反映了不同的特征之间的相互关系。比如在原始图像上只有3个特征“红·绿·蓝”。它们的Gram矩阵就是一个 3×3 的矩阵，反映了图像在颜色上的某种“风格”。

构造优化问题

能够“量化”地表示“风格”，剩下来就好办了。用 Y 记录要处理的照片，用 X 变量记录最终要输出的结果，用 $I^{\ell_c}(\cdot)$ 代表直到 ℓ_c 层的复合变换。 ℓ_c 是我们选定的能够表现内容的层的编号。

内容的Loss

$$L_{\text{content}}(X, Y) = \frac{1}{2} \|I^{\ell_c}(X) - I^{\ell_c}(Y)\|^2$$

用 Z 记当作风格的图片（如《星空》），用 $G^{\ell_s}(\cdot)$ 代表由 ℓ_s 层计算Gram矩阵的映射， w_s 代表这层在风格中的重要性。

Style的Loss

$$L_{\text{style}}(X, Z) = \sum_s w_s \|G^{\ell_s}(X) - G^{\ell_s}(Z)\|^2$$

最终要优化的目标就是

Loss函数

$$L(X, Y, Z) = \alpha L_{\text{content}}(X, Y) + \beta L_{\text{style}}(X, Z)$$

这里 α/β 大概在 $1 \times 10^{-4} \sim 1 \times 10^{-3}$ 的数量级。

最终融合 Z 的Style和 Y 的内容的图像 X

问题最终转化为从一个随机的初值 X_0 开始，求解下面优化问题

$$\min_X L(X, Y, Z)$$

实践上，可以走**Caffe**路线，手工推导**Loss**的梯度，然后用**Caffe**的反向方法计算更新量，迭代求解。或是**TensorFlow**风格，用原来网络中的张量构造出上面的**Loss**张量，然后调用优化器求解。

用思想武装大脑，用工具武装身体

总结一下这个例子中解决问题的“哲学”

哲学

思想 建模的思想

- 想办法把一个要解决的问题转换为一个优化问题
- 求解这个优化问题

工具 DeepLearning的算法和框架

- BP算法
- TensorFlow或其他的框架

再回首—“漫长的”三周

第一讲

从机器学习的基本思路开始，讲解了神经网络模型的组成方式。详细推导了向后传播算法的公式，和几个具体情况下中间层和Loss层的特殊形式。计算了卷积层的反向公式，得到卷积层反向仍然是卷积的结论。

第二讲

紧接着以Caffe为例，剖析了现代深度学习库的框架，解释了常用优化器的实现原理。另一方面以TensorFlow为例，解释了DL框架的另一个流派—Computational Graph的编程思路。

最后一讲

运用学过的概念，解释了和DL相关的有趣应用。

一家之言：“深度学习”是什么？

“一千个人眼中有一千个哈姆雷特”

和大家一起看了这么多之后，不妨讲讲我的一些粗浅理解：

- 表示学习
- 自动的特征抓取
- 浅层表示可以被后续层构造成抽象的概念，“概念学习”
- 即使是为了某个特殊目的训练的网络，其学到的特征却是通用的，可以用于其他的任务
- 每个组件都能用可微分的映射表示，从而能使用BP算法训练。从框架级别来说，容易构造端到端的应用

未来



