**深度学习基础**

1. **深度学习概述**
   1. **深度学习的引入**

|  |
| --- |
|  |

* **深度学习(Deep leaning):**是一种实现机器学习的技术，是机器学习重要分支。其源于人工神经网络的研究。深度学习的模型结构是一种含多隐层的神经网络。通过组合低层特征形成更加抽象的高层特征。

|  |
| --- |
|  |

* 1. **数据集及其拆分**
* **数据集(dataset)的数学表示**

|  |
| --- |
|  |

* **类别标签的ground truth与gold standard**

|  |
| --- |
|  |

* **数据集与有监督学习**

|  |
| --- |
|  |

* **训练集、测试集的拆分**

|  |
| --- |
|  |

|  |
| --- |
|  |

* **用网络搜索来调超参数**

|  |
| --- |
|  |

* 1. **分类及其性能度量**

|  |
| --- |
|  |

* **分类性能度量——准确率**
* 假设只有两类样本，即正例(positive)和负例(negative)。通常以关注的类为正类，其他类为负类。

|  |
| --- |
|  |

**表中AB模式：第一个符号表示预测结果对了(True)还是错了(False),第二个符号表示预测的类别。**

* 分类准确率(accuracy):分类器正确分类的样本数与总样本数之比：



* **分类性能度量——精确率和召回率**

|  |
| --- |
|  |

* **精确率（precision）和召回率（recall):**是二分类问题常用的评价指标。

****

* **精确率**反映了模型**判定的正例**中真正正例的比重。在垃圾短信分类器中，是指预测出的垃圾短信中真正垃圾短信的比例。
* **召回率**反应**总正例**中被模型正确判定正例的比重。医学领域也叫灵敏度(sensitivity)。在垃圾短信分类器中，指所有真的垃圾短信被分类器被正确找出来的比例。
* **分类性能度量——P-R曲线**

|  |
| --- |
|  |

* **Area(Area Under Curve,简称AUC)**
* Area的定义（p-r曲线下的面积）如下：



* + Area有助于弥补P、R的单点值局限性，可以反应全局性能。
* **如何绘制P-R曲线**

|  |
| --- |
|  |

* **分类性能度量——F值**
* F值是精确率和召回率的调和平均：



* 一般大于0。当时，退化为：



* 比较常用的是，即表示二者同等重要。
* **分类性能度量——ROC**

|  |
| --- |
|  |

* **横轴：**假正例率
* **纵轴：**真正例率
* **（**受试者工作特征曲线，receiver operating characteristic curve)描绘了分类器在（真正正例占总正例的比率，反映命中率，纵轴）和(错误的正例占反例的比率，反映误诊率、假阳性率，横轴）间的trade-off。
* **分类性能度量——ROC曲线绘制**

|  |
| --- |
|  |

* **分类性能度量——ROC-AUC计算**

|  |
| --- |
|  |

* （Area Under Curve)定义为ROC曲线下的面积。
* AUC值提供了分类器的整体数值。通常AUC越大，分类更好。
* 取值范围为[0,1]
* **分类报告**
* **分类报告（Classification report)**显式每个类的分类性能。包括每个类标签的精确率、召回率、F1值等。

|  |
| --- |
|  |

* 1. **回归问题极其性能评价**
* **什么是回归**

回归分析（regression analysis）是确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。和分类问题不同，回归通常输出为一个实数数值。而分类的输出通常为若干指定的类别标签。

|  |
| --- |
|  |

* **回归性能度量方法（Regression metrics）**
* **常用的评价回归问题的方法：**

1. 平均绝对误差MAE(mean\_absolute\_error)
2. 均方误差MSE(mean\_squared\_error)及均方根差RMSE
3. Log loss, 或称交叉熵loss(cross-entropy loss)
4. R方值，确定系数（r2\_score) (后文介绍)

* **评价绝对误差MAE**

|  |
| --- |
|  |

* **均方误差MSE**

|  |
| --- |
|  |

* **logistic回归损失，简称Log loss,或交叉熵损失（cross-entropy loss）**
* 常用于评价逻辑回归LR和神经网络。
* **对于二类分类问题：**

1）假设某样本的真实标签为y(取值为0或1)，概率估计为

2）每个样本的log loss是对分类器给定真实值标签的负log似然估计(negative log-likelihood):



3）例子：

|  |
| --- |
|  |

* + **对于多类分类问题（multiclass problem）：**可将样本的真实标签（true label）编码成1-of-K(K为类别总数)的二元指示矩阵Y :

|  |
| --- |
|  |

假设模型对测试样本的概率估计结果为P,则在测试集上（假设测试样本总数为N）

上的交叉熵损失表示如下：



其中表示第i个样本的第k个标签的真实值，注意由于表示为1-of-K模式，

因此每个样本只有其中一个标签值为1，其余均为0。表示模型对该样本的预 测值。

|  |
| --- |
|  |

* 1. **一致性的评价方法**
* **一致性评价：**是指对两个或多个相关的变量进行分析，从而衡量其相关性的密切程度。
* **一致性评价——皮尔森(pearson)相关系数法**

|  |
| --- |
|  |

* **一致性评价——Cohen’s kappa 相关系数**
* Cohen’s kappax相关系数也可用于衡量两个评价者之间的一致性。其特点在于：
* 与pearson相关系数的区别：Cohen's kappa相关系数**通常用于离散的**分类的一致性评价。
* 其通常被认为比两个人之间的简单一致百分比更强壮，因为Cohen’s kappa考虑到了二人之间的随机一直的可能性。
* 如果评价者多于2人时，可以考虑使用Fleiss’ kappa.
* **Cohen’s kappa计算方法**

|  |
| --- |
|  |
|  |

* **Fleiss’s kappa计算方法**

|  |
| --- |
|  |

* 1. **程序：使用Sklean进行精确率-召回率曲线的绘制**

|  |
| --- |
| 1. #coding=utf-8 2. """ 3. #演示目的：利用鸢尾花数据集画出P-R曲线 4. """ 5. print(\_\_doc\_\_) 6. **import** matplotlib.pyplot as plt 7. **import** numpy as np 8. **from** sklearn **import** svm, datasets 9. **from** sklearn.metrics **import** precision\_recall\_curve 10. **from** sklearn.metrics **import** average\_precision\_score 11. **from** sklearn.preprocessing **import** label\_binarize 12. **from** sklearn.multiclass **import** OneVsRestClassifier 13. #from sklearn.cross\_validation import train\_test\_split  #适用于anaconda 3.6及以前版本 14. **from** sklearn.model\_selection **import** train\_test\_split#适用于anaconda 3.7 15. #以iris数据为例，画出P-R曲线 16. iris **=** datasets.load\_iris() 17. X **=** iris.data 18. y **=** iris.target 19. # 标签二值化,将三个类转为001, 010, 100的格式.因为这是个多类分类问题，后面将要采用 20. #OneVsRestClassifier策略转为二类分类问题 21. y **=** label\_binarize(y, classes**=**[0, 1, 2]) 22. n\_classes **=** y.shape[1] 23. print (y) 24. # 增加了800维的噪声特征 25. random\_state **=** np.random.RandomState(0) 26. n\_samples, n\_features **=** X.shape 27. X **=** np.c\_[X, random\_state.randn(n\_samples, 200 **\*** n\_features)] 28. # Split into training and test 29. X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**.5, random\_state**=**random\_state) #随机数，填0或不填，每次都会不一样 30. # Run classifier probability : boolean, optional (default=False)Whether to enable probability estimates. This must be enabled prior to calling fit, and will slow down that method. 31. classifier **=** OneVsRestClassifier(svm.SVC(kernel**=**'linear', probability**=**True, random\_state**=**random\_state)) 32. y\_score **=** classifier.fit(X\_train, y\_train).decision\_function(X\_test) 33. # Compute Precision-Recall and plot curve 34. #下面的下划线是返回的阈值。作为一个名称：此时“\_”作为临时性的名称使用。 35. #表示分配了一个特定的名称，但是并不会在后面再次用到该名称。 36. precision **=** dict() 37. recall **=** dict() 38. average\_precision **=** dict() 39. **for** i **in** range(n\_classes): 40. precision[i], recall[i], \_ **=** precision\_recall\_curve(y\_test[:, i],  y\_score[:, i]) #The last precision and recall values are 1. and 0. respectively and do not have a corresponding threshold. This ensures that the graph starts on the x axis. 41. average\_precision[i] **=** average\_precision\_score(y\_test[:, i], y\_score[:, i])#切片，第i个类的分类结果性能 42. # Compute micro-average curve and area. ravel()将多维数组降为一维 43. precision["micro"], recall["micro"], \_ **=** precision\_recall\_curve(y\_test.ravel(),  y\_score.ravel()) 44. average\_precision["micro"] **=** average\_precision\_score(y\_test, y\_score, average**=**"micro") #This score corresponds to the area under the precision-recall curve. 45. # Plot Precision-Recall curve for each class 46. plt.clf()#clf 函数用于清除当前图像窗口 47. plt.plot(recall["micro"], precision["micro"], 48. label**=**'micro-average Precision-recall curve (area = {0:0.2f})'.format(average\_precision["micro"])) 49. **for** i **in** range(n\_classes): 50. plt.plot(recall[i], precision[i], 51. label**=**'Precision-recall curve of class {0} (area = {1:0.2f})'.format(i, average\_precision[i])) 52. plt.xlim([0.0, 1.0]) 53. plt.ylim([0.0, 1.05]) #xlim、ylim：分别设置X、Y轴的显示范围。 54. plt.xlabel('Recall', fontsize**=**16) 55. plt.ylabel('Precision',fontsize**=**16) 56. plt.title('Extension of Precision-Recall curve to multi-class',fontsize**=**16) 57. plt.legend(loc**=**"lower right")#legend 是用于设置图例的函数 58. plt.show() |

1. **特征工程概述**

**2.1 特征工程**

|  |
| --- |
|  |

**2.2 向量空间模型及文本相似度计算**

* **文档的向量化表示：BOW假设和VSM模型**

|  |
| --- |
|  |

* **停用词**

|  |
| --- |
|  |

* **N-gram模型**

|  |
| --- |
|  |

* **文档之间的欧式距离**

|  |
| --- |
|  |

* **文档之间的余弦相似度**

|  |
| --- |
|  |

* **Tf-idf词条权重计算**

|  |
| --- |
|  |

|  |
| --- |
|  |

**2.3 特征处理（特征缩放、选择及降维）**

* **特征值的缩放**

|  |
| --- |
|  |

* **特征值的缩放——标准化法**
* 标准化的前提是特征值服从正态分布。
* 标准化需要计算特征值的均值和标准差，公式表达为：



* **标准差:**
* 标准差（standard Deviation），常称均方差，用表示，是方差的算术平方根。



* + 标准差能反映一个数据集的离散程度。
* **特征值的缩放——区间缩放法**
* 区间缩放法利用了边界值信息，将特征的取值区间缩放到某个特定范围。假设max和min为希望的调整后范围，则



* 由于希望的调整后范围一般为[0,1]。此时，公式变为



* **特征值的归一化**

|  |
| --- |
|  |

* **定量特征的二值化**

|  |
| --- |
|  |

* **缺失特征值的弥补计算**

|  |
| --- |
|  |

* **创建多项式特征**

|  |
| --- |
|  |

* **特征选择**

|  |
| --- |
|  |

* **特征选择方法1——方差选择法**

|  |
| --- |
|  |

* **特征选择方法2——皮尔森相关系数法**

|  |
| --- |
|  |

* **特征选择方法3——基于森林的特征选择**

|  |
| --- |
|  |

* **特征选择方法4——递归特征消除法**

|  |
| --- |
|  |

* **特征降维——线性判别分析法（LDA）**

|  |
| --- |
|  |

* **特征降维——主成成分分析法（PCA）**

|  |
| --- |
|  |

**2.4 使用Sklean对文档进行向量化**

|  |
| --- |
| 1. # -\*- coding: utf-8 -\*- 2. """ 3. 演示内容：文档的向量化 4. """ 5. **from** sklearn.feature\_extraction.text **import** CountVectorizer 6. corpus **=** [ 7. 'Jobs was the chairman of Apple Inc., and he was very famous', 8. 'I like to use apple computer', 9. 'And I also like to eat apple' 10. ] 12. #未经停用词过滤的文档向量化 13. vectorizer **=**CountVectorizer() 14. print(vectorizer.fit\_transform(corpus).todense())  #转化为完整特征矩阵 15. print(vectorizer.vocabulary\_) 17. print(" ")  20. #经过停用词过滤后的文档向量化 21. **import** nltk 22. nltk.download('stopwords') 23. stopwords **=** nltk.corpus.stopwords.words('english') 24. print (stopwords) 25. print(" ") 26. vectorizer **=**CountVectorizer(stop\_words**=**'english') 27. print("after stopwords removal:  ", vectorizer.fit\_transform(corpus).todense()) 28. print("after stopwords removal:  ", vectorizer.vocabulary\_) 30. print(" ") 31. #采用ngram模式进行文档向量化 32. vectorizer **=**CountVectorizer(ngram\_range**=**(1,2))#表示从1-2，既包括unigram，也包括bigram 33. print("N-gram mode:     ",vectorizer.fit\_transform(corpus).todense())  #转化为完整特征矩阵 34. print(" ") 35. print("N-gram mode:         ",vectorizer.vocabulary\_) |

**2.5 使用Sklean进行量纲缩放**

|  |
| --- |
| 1. # -\*- coding: utf-8 -\*- 2. """ 3. 演示内容：量纲的特征缩放 4. （两种方法：标准化缩放法和区间缩放法。每种方法举了两个例子：简单二维矩阵和iris数据集） 5. """ 6. #方法1：标准化缩放法 例1：对简单示例二维矩阵的列数据进行 7. **from** sklearn **import** preprocessing 8. **import** numpy as np 9. #采用numpy的array表示，因为要用到其mean等函数，而list没有这些函数 10. X **=** np.array([[0, 0], 11. [0, 0], 12. [100, 1], 13. [1, 1]]) 14. # calculate mean 15. X\_mean **=** X.mean(axis**=**0) 16. # calculate variance 17. X\_std **=** X.std(axis**=**0) 18. #print (X\_std) 19. # standardize X 20. X1 **=** (X**-**X\_mean)**/**X\_std 21. print (X1) 22. print ("") 24. # we can also use function preprocessing.scale to standardize X 25. X\_scale **=** preprocessing.scale(X) 26. print (X\_scale)  29. #方法1： 标准化缩放法 例2：对iris数据二维矩阵的列数据进行。这次采用一个集成的方法StandardScaler 30. **from** sklearn **import** datasets 31. iris **=** datasets.load\_iris() 32. X\_scale **=** preprocessing.scale(iris.data) 33. print (X\_scale) 35. #方法2： 区间缩放法 例3：对简单示例二维矩阵的列数据进行 36. **from** sklearn.preprocessing **import** MinMaxScaler 38. data **=** [[0, 0], 39. [0, 0], 40. [100, 1], 41. [1, 1]] 43. scaler **=** MinMaxScaler() 44. print(scaler.fit(data)) 45. print(scaler.transform(data)) 47. #方法2： 区间缩放法 例4：对iris数据二维矩阵的列数据进行 48. **from** sklearn.preprocessing **import** MinMaxScaler 50. data **=** iris.data 52. scaler **=** MinMaxScaler() 53. print(scaler.fit(data)) 54. print(scaler.transform(data)) |

1. **回归问题及正则化**

**3.1 线性回归模型及其求解方法**

* **回归和分类的区别和联系**

|  |
| --- |
|  |

* **线性模型**
* **狭义线性（linear）模型：**
* 通常指自变量与因变量之间按比例、成直线的关系，在数学上可理解为一阶导数为常数的函数，如；
* 线性通常表现为一次曲线。
* **广义线性（generalized linear model,GLM）：**
* 是线性模型的扩展，主要通过联结函数g()(link function),是预测值落在响应变量的变幅内。例如逻辑回归

 （括号内为线性函数）

* **非线性模型**

|  |
| --- |
|  |

* **线性回归**
* 线性回归模型中，假设自变量和因变量满足如下形式：



* 问题：已知一些数据，如何求里面的未知参数，给出一个最优解。
* 因此通过将求解问题转化为求最小误差问题。一般采用模型预测结果与真是结果的差的平方和作为损失函数：



即求

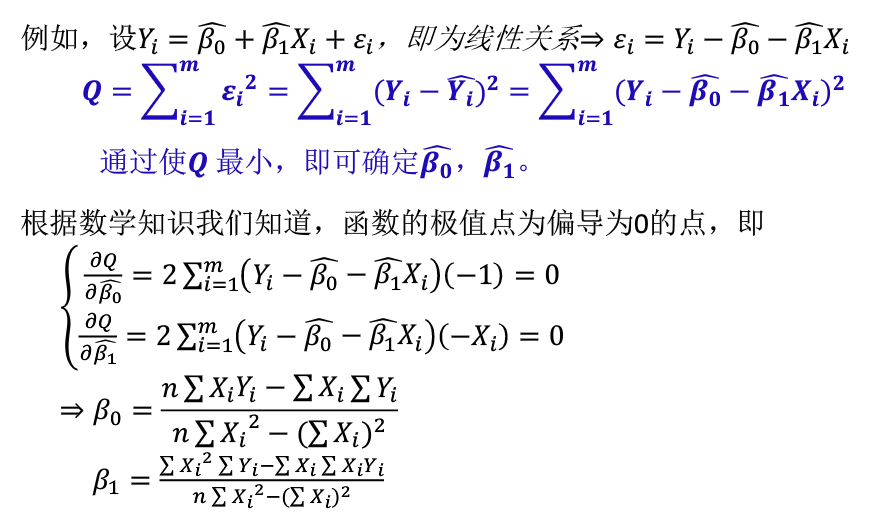
* **概率解释**
* 设预测结果与真实结果之间误差为，即。
* 通常误差满足平均值为0的高斯分布，即正太分布。那么在一个样本i上x和y的概率密度公式为
* 模型在全部样本上预测的最大似然估计为：





从而，需要

* **求解参数**
* 接下来，就是求解使得最小的参数。
* 解法有：
* 矩阵解法。Scikit-learn中的LinearRegression类使用的是矩阵解法（有时也称为最小二乘法）。可以解出线性回归系数。



* 梯度下降。梯度下降（Gradient descent）是利用一阶的梯度信息找到函数局部最优解的一种方法。

|  |
| --- |
|  |

**3.2 多元回归与多项式回归**

* **Sklean的一元线性回归**

|  |
| --- |
|  |

* **残差（residual）**

|  |
| --- |
|  |

* **多元线性回归**

|  |
| --- |
|  |

* **R方计算步骤**

|  |
| --- |
|  |

* **多项式回归**

|  |
| --- |
|  |

**3.3 损失函数的正则化**

* **向量范数**

**向量范数，假设向量有*N*个元素。**

* **L1-范数：即向量元素绝对值之和。**

****

* **L2-范数：Eucilid范数（欧几里得范数，常用计算向量长度），即向量元素绝对值的平方和再开方。**

****

* **范数：即所有向量元素绝对值中的最大值。**

****

* **P-范数：即向量元素绝对值的P次方和的1/p次幂。**

****

* **矩阵范数**

**假设矩阵A为m\*n,即m行，n列。**

* **L1-范数：列和范数，即矩阵的所有列向量元素绝对值之和的最大值。**

****

* **L2-范数：谱范数，即最大特征值的开平方。**

**，最大特征值。**

* **行和范数：即矩阵的所有行向量元素绝对值之和的最大值。**

****

* **F-范数：Frobenius范数，即矩阵元素绝对值的平方和再开方。**

****

* **线性回归的正则化**

|  |
| --- |
|  |

* **线性回归的正则化后的梯度更新方法**

|  |
| --- |
|  |

|  |
| --- |
|  |

**3.4 逻辑回归**

* **逻辑回归（Logistic Regression）**

|  |
| --- |
|  |

* **逻辑回归中的损失函数优化方法**

|  |
| --- |
|  |

* **坐标下降法**

|  |
| --- |
|  |

**3.5 程序讲解：使用sklearn进行线性回归和二次回归**

|  |
| --- |
| 1. #coding=utf-8 2. """ 3. #演示内容：二次回归和线性回归的拟合效果的对比 4. """ 5. print(\_\_doc\_\_) 7. **import** numpy as np 8. **import** matplotlib.pyplot as plt 9. **from** sklearn.linear\_model **import** LinearRegression 10. **from** sklearn.preprocessing **import** PolynomialFeatures 11. **from** matplotlib.font\_manager **import** FontProperties 12. font\_set **=** FontProperties(fname**=**r"c:\windows\fonts\simsun.ttc", size**=**20) 14. **def** runplt(): 15. plt.figure()# 定义figure 16. plt.title(u'披萨的价格和直径',fontproperties**=**font\_set) 17. plt.xlabel(u'直径（inch）',fontproperties**=**font\_set) 18. plt.ylabel(u'价格（美元）',fontproperties**=**font\_set) 19. plt.axis([0, 25, 0, 25]) 20. plt.grid(True) 21. **return** plt  24. #训练集和测试集数据 25. X\_train **=** [[6], [8], [10], [14], [18]] 26. y\_train **=** [[7], [9], [13], [17.5], [18]] 27. X\_test **=** [[7], [9], [11], [15]] 28. y\_test **=** [[8], [12], [15], [18]] 30. #画出横纵坐标以及若干散点图 31. plt1 **=** runplt() 32. plt1.scatter(X\_train, y\_train,s**=**40) 34. #给出一些点，并画出线性回归的曲线 35. xx **=** np.linspace(0, 26, 5) 36. regressor **=** LinearRegression() 37. regressor.fit(X\_train, y\_train) 38. yy **=** regressor.predict(xx.reshape(xx.shape[0], 1)) 40. plt.plot(xx, yy, label**=**"linear equation") 42. #多项式回归（本例中为二次回归） 43. #首先生成多项式特征 44. quadratic\_featurizer **=** PolynomialFeatures(degree**=**2) 45. X\_train\_quadratic **=** quadratic\_featurizer.fit\_transform(X\_train) 47. regressor\_quadratic **=** LinearRegression() 48. regressor\_quadratic.fit(X\_train\_quadratic, y\_train) 50. #numpy.reshape（重塑）给数组一个新的形状而不改变其数据。在指定的间隔内返回均匀间隔的数字 51. #给出一些点，并画出线性回归的曲线 52. xx **=** np.linspace(0, 26, 5) 53. print (xx.shape) 54. print (xx.shape[0]) 55. xx\_quadratic **=** quadratic\_featurizer.transform(xx.reshape(xx.shape[0], 1)) 56. print (xx.reshape(xx.shape[0], 1).shape) 58. plt.plot(xx, regressor\_quadratic.predict(xx\_quadratic), 'r-',label**=**"quadratic equation") 59. plt.legend(loc**=**'upper left') 60. plt.show() 62. X\_test\_quadratic **=** quadratic\_featurizer.transform(X\_test) 63. print('linear equation  r-squared', regressor.score(X\_test, y\_test)) 64. print('quadratic equation r-squared', regressor\_quadratic.score(X\_test\_quadratic, y\_test)) |

1. **信息熵及梯度计算**

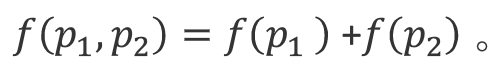
**4.1信息熵（Information Entropy）**

* **信息论中的熵**

|  |
| --- |
|  |

* **信息熵**
* 信源信息的不确定性函数*f*通常满足两个条件：

1. 是概率的单调递减函数。
2. 两个独立符号所产生的不确定性应等于各自不确定性之和，即



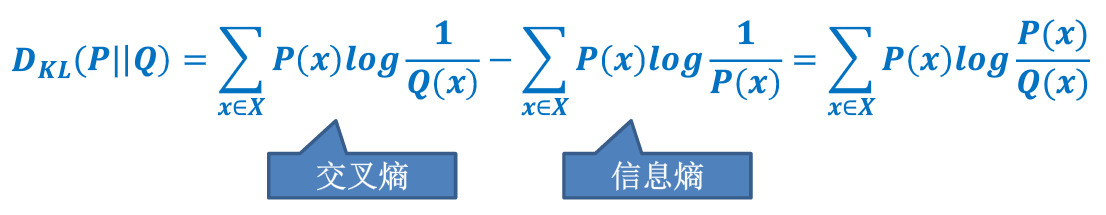
* 对数函数同时满足这两个条件：
* 信息熵：要考虑信源所有可能发生情况的平均不确定性。若信源符号有n种取值：对应概率为且各种出现彼此独立。此时信源的平均不确定性应当为单个符号不确定性的统计平均值（E），称为信息熵，即



* **交叉熵（cross entropy）**
* 定义：交叉熵用于表征两个变量概率分布P、Q(假设P表示真实分布，Q为模型预测的分布）的差异性。交叉熵越大，两个变量差异程度越大。
* 交叉熵公式：



* **相对熵（relative entropy）**
* 也称为KL散度（Kullback-Leibler divergence,简称KLD）、信息散度（information divergence）、信息增益（information gain）。
* **相对熵的定义**：是交叉熵与信息熵的差值。表示分布Q模拟真实分布P,所需的额外信息。
* **计算公式为：**



* **相对熵举例**

|  |
| --- |
|  |

* **相对熵的性质**

|  |
| --- |
|  |

* **JS散度**

|  |
| --- |
|  |

* **联合熵**
* 联合熵（复合熵，Joint Entropy）
* 用H(X,Y)表示
* 两个随机变量X,Y 的联合分布的熵，形成联合熵。
* **条件熵**
* 条件熵（the conditional entropy）：H(X|Y)表示在已知随机变量Y的条件下随机变量X的不确定性。
* H(X|Y)=H(X,Y)-H(Y) 表示（X,Y）的联合熵，减去Y单独发生包含的熵。

推导过程：

|  |
| --- |
|  |

* **互信息**
* 互信息（Mutual Information）可以被看成是一个随机变量中包含的关于另一个随机变量的信息量，或者说是一个随机变量由于已知另一个随机变量而减少的不确定性。

****

即互信息是联合发布与乘积分布的相对熵。

* **文氏图图解**

|  |
| --- |
|  |

**4.2 反向传播中的梯度（Gradient in Backpropagation）**

* **反向传播中的梯度**

|  |
| --- |
|  |

* **梯度的简单解释**

|  |
| --- |
|  |

|  |
| --- |
|  |

* **矩阵-矩阵相乘的梯度**

|  |
| --- |
|  |

**4.3 感知机**

* **什么是感知机（perception）**

|  |
| --- |
|  |

* **感知机模型的损失函数**

|  |
| --- |
|  |

* 损失函数的选择采用误分类点到超平面的距离（这里采用点到直线的距离）：

，其中是L2范数。

* 对于误分类点来说：总是成立。则误分类点到超平面的距离可以写为，从而所有分类点到超平面的总距离为。
* 不考虑，就得到极小化损失函数。其中M为误分类点的集合。
* **感知机模型的优化——随机梯度下降法**
* 优化目的：找到是损失函数变小的参数。
* 模型参数，即w和b的更新公式为

 （其中是步长，0<1，是梯度）

对w和b求的两个偏导分别为：

感知机算法的损失函数极小化过程是每次随机选择一个误分类点使其梯度下降。因此，随机选择的一个误分类点，对w和b进行更新。即

， 

* **感知机学习算法**

|  |
| --- |
|  |

* **感知机迭代实例**

|  |
| --- |
|  |

**4.4 正向传播和反向传播的程序示例**

|  |
| --- |
|  |

**4.5 信息熵和互信息的计算程序示例**

|  |
| --- |
| 1. # -\*- coding: utf-8 -\*- 2. #演示内容：香农信息熵的计算(例1和例2分别为两种不同类型的输入)以及互信息的计算（例3）。其中log默认为自然对数。 4. **import** numpy as np 5. **from** math **import** log 7. #例1： 计算香农信息熵（已知概率分布） 8. print("例1：") 9. **def** calc\_ent(x): 10. ent **=** 0.0 11. **for** p **in** x: 12. ent **-=** p **\*** np.log(p) 13. **return** ent 15. x1**=**np.array([0.4, 0.2, 0.2, 0.2]) 16. x2**=**np.array([1]) 17. x3**=**np.array([0.2, 0.2, 0.2, 0.2, 0.2]) 18. print ("x1的信息熵:", calc\_ent(x1)) 19. print ("x2的信息熵:", calc\_ent(x2)) 20. print ("x3的信息熵:", calc\_ent(x3)) 21. print("") 23. #例2： 计算香农信息熵（此时给定了信号发生情况) 24. print("例2：") 25. **def** calcShannonEnt(dataSet): 26. length,dataDict**=**float(len(dataSet)),{} 27. **for** data **in** dataSet: 28. **try**:dataDict[data]**+=**1 29. **except**:dataDict[data]**=**1 30. **return** sum([**-**d**/**length**\***log(d**/**length) **for** d **in** list(dataDict.values())]) 31. print("x1的信息熵:", calcShannonEnt(['A','B','C','D','A'])) 32. print("x2的信息熵:",calcShannonEnt(['A','A','A','A','A'])) 33. print("x3的信息熵:",calcShannonEnt(['A','B','C','D','E']))  36. #例3： 计算互信息（输入：给定的信号发生情况,其中联合分布已经手工给出) 37. print("") 38. print("例3：") 39. Ent\_x4**=**calcShannonEnt(['3',  '4',   '5',  '5', '3',  '2',  '2', '6', '6', '1']) 40. Ent\_x5**=**calcShannonEnt(['7',  '2',   '1',  '3', '2',  '8',  '9', '1', '2', '0']) 41. Ent\_x4x5**=**calcShannonEnt(['37', '42', '51', '53', '32', '28', '29', '61', '62', '10', '37', '42']) 42. MI\_x4\_x5**=**Ent\_x4**+**Ent\_x5**-**Ent\_x4x5 43. print ("x4和x5之间的互信息:",MI\_x4\_x5) |

1. **循环神经网络及其变体**

**5.1 循环神经网络（Recurrent Neural Network）**

* **什么是循环神经网络**

|  |
| --- |
|  |

* **RNN序列处理**

|  |
| --- |
|  |

* **最基本的RNN结构**

|  |
| --- |
|  |

* **基本RNN的计算过程**

|  |
| --- |
|  |

* **RNN的参数共享**

|  |
| --- |
|  |

**5.2 长短时记忆网络（Long Short-Term Memory）**

* **标准的RNN可以处理短期依赖**

|  |
| --- |
|  |

* **标准的RNN难以应付长期依赖**

|  |
| --- |
|  |

* **LSTM的基本思路**

|  |
| --- |
|  |

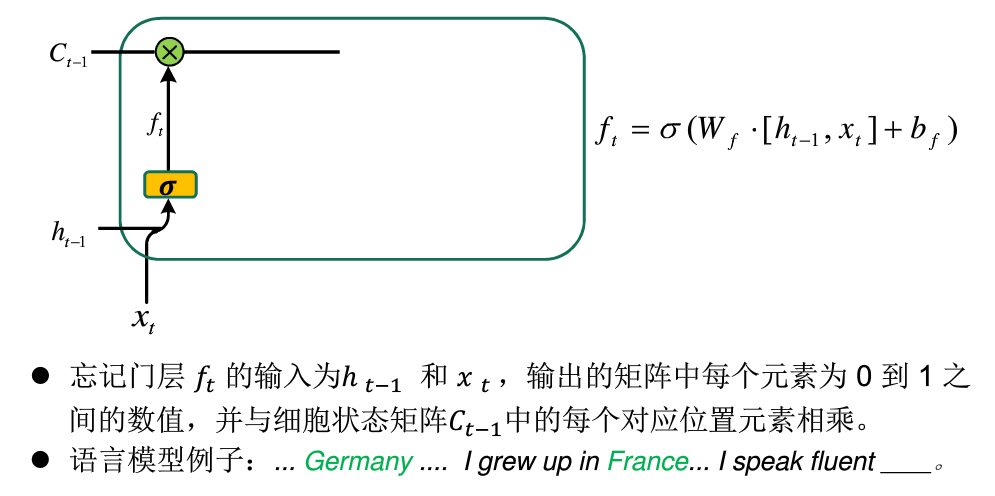
* **神经网络中的门**

|  |
| --- |
|  |

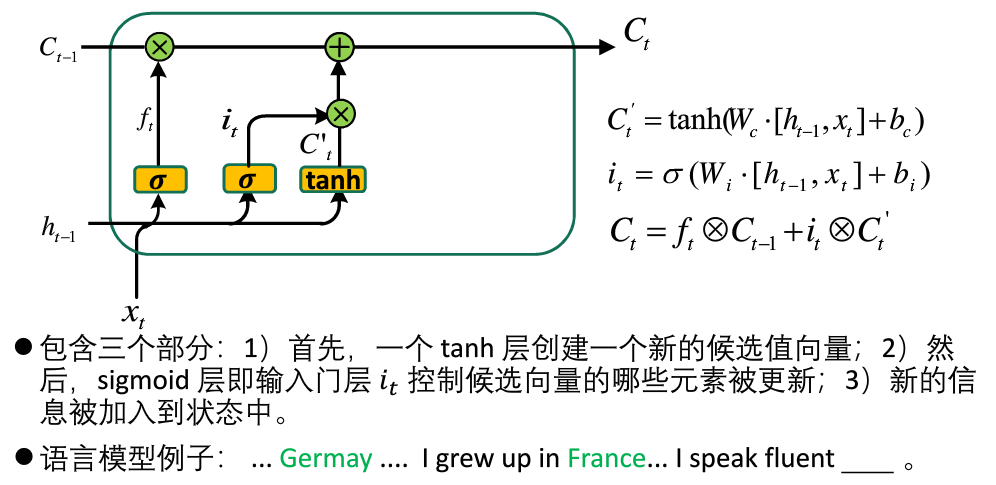
* **LSTM的整体框架图**

|  |
| --- |
|  |

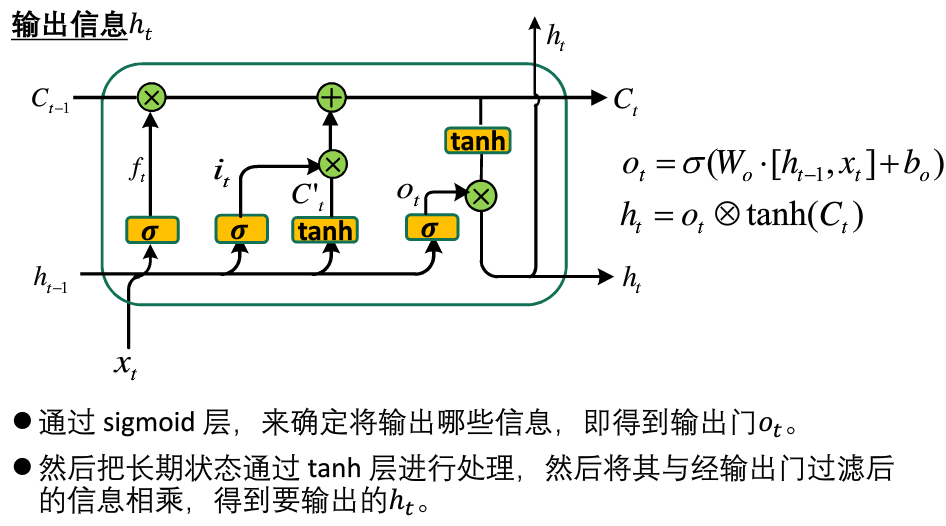
* **LSTM的计算过程**
* **忘记信息：**从长期状态中丢弃某些信息。



* **新记忆信息：**将新信息存放在长期状态中。



* **输出信息**



* **LSTM的变体（1）**

|  |
| --- |
|  |

* **LSTM的变体（2）**

|  |
| --- |
|  |

* **LSTM的变体（3）——GRU**

|  |
| --- |
|  |

**5.3 双向循环神经网络和注意力**

* **双向RNN(Bidirectional RNNs)**

|  |
| --- |
|  |

* **深层双向RNN(Deep Bidirectional RNNs)**

|  |
| --- |
|  |

* **注意力模型**

|  |
| --- |
|  |

* **注意力模型基本原理**

|  |
| --- |
|  |

**5.4 循环神经网络的程序示例**

|  |
| --- |
| 1. #演示内容：利用tensorflow实现的循环神经网络RNN（本程序使用了LSTM）来做语言模型，并输出其困惑度。 2. #语言模型主要是根据一段给定的文本来预测下一个词最有可能是什么。困惑度用于评价语言模型。困惑度越小，则模型的性能越好。 4. **import** tensorflow as tf 6. #reader模块用于读取数据集，以及做训练集、验证集和测试集的切分等。如果无法导入reader模块，也可以从网上下载reader.py，加入到同级目录即可，下载地址：https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/reader.py 7. **import** reader 8. **import** numpy as np 10. DATA\_PATH**=**'data' #data目录中应该预先存放本程序中要用到的语料，即PTB(Penn Treebank Dataset)数据集，也就是宾州树库数据集 。PTB数据集是语言模型研究常用的数据集。其下载地址在 Tomas Mikolov的主页： https://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz。解压后，将其中的整个data文件夹拷贝到当前目录即可。数据集共有9998个单词，加上稀有词语的特殊符号<unk>和语句的结束标记，共有10000个单词。 12. #超参数的设置 13. HIDDEN\_SIZE**=**200#size of RNN hidden state，指表示每个单词的词向量的维度 14. NUM\_LAYERS**=**2   #表示使用了两个LSTM层 15. VOCAB\_SIZE**=**10000 #词汇表中词的个数 17. LEARNING\_RATE**=**1.0 # 设置学习速率超参数 18. TRAIN\_BATCH\_SIZE**=**20  # 训练阶段每个数据批量设置为多少个样本。在本例中每个样本是指由若干个词构成的一个序列 19. TRAIN\_NUM\_STEP**=**35    # 训练阶段时间步的个数，即在训练阶段中文本数据的截断长度，也可以称为序列长度seq\_length 21. # 可以将测试数据看成一个很长的序列，所以下面这两个参数都设置为1 22. EVAL\_BATCH\_SIZE**=**1    #在后面main函数中计算valid\_batch\_len时会用到 23. EVAL\_NUM\_STEP**=**1      #在后面main函数中计算valid\_epoch\_size时会用到 24. NUM\_EPOCH**=**2    #训练的轮数 25. KEEP\_PROB**=**0.5  # 节点不被dropout的概率 26. MAX\_GRAD\_NORM**=**5 #这个超参数后面会用到，用于避免梯度膨胀问题 28. # 类PTBModel的定义，是这个多层循环神经网络模型的描述 29. **class** PTBModel(object): 30. #初始化函数，其参数包括当前是否在训练阶段，批量的大小，数据的截断长度即序列长度等 31. **def** \_\_init\_\_(self,is\_training,batch\_size,num\_steps): 32. # 根据给定的参数数值来设置使用的batch大小和时间步数（截断长度） 33. self.batch\_size**=**batch\_size 34. self.num\_steps**=**num\_steps 36. # 定义输入层的数据维度为: batch\_size \* num\_steps。 这里tf.placeholder可以理解为采用占位符进行占位，等以后实际运行时进行具体填充。等建立session，运行模型的时候才feed数据 37. self.input\_data**=**tf.placeholder(tf.int32,[batch\_size,num\_steps]) 39. # 定义输出层的数据维度为：batch\_size\*num\_steps。num\_steps相当于seq\_length 40. self.targets**=**tf.placeholder(tf.int32,[batch\_size,num\_steps]) 42. # 定义使用LSTM结构作为循环体的基本结构。每个单词向量的维度为HIDDEN\_SIZE 43. lstm\_cell**=**tf.nn.rnn\_cell.BasicLSTMCell(HIDDEN\_SIZE) 44. #如果是在训练阶段，则使用dropout.此时每个单元以（1-keep\_prob）的概率不工作，目的是防止过拟合。 45. **if** is\_training: 46. lstm\_cell**=**tf.nn.rnn\_cell.DropoutWrapper(lstm\_cell,output\_keep\_prob**=**KEEP\_PROB) 48. #将多层RNN单元封装到一个单元cell中。层的个数NUM\_LAYERS前面已经设定为2了 49. cell**=**tf.nn.rnn\_cell.MultiRNNCell([lstm\_cell]**\***NUM\_LAYERS) 51. # 使用zero\_state函数来初始化网络的状态 52. self.initial\_state**=**cell.zero\_state(batch\_size,tf.float32) 54. # 下面两条语句是将单词转换成向量 55. #VOCAL\_SIZE即单词的总数；  HIDDEN\_SIZE，即每个单词的向量维度。所以 embedding参数的维度为VOCAB\_SIZE \* HIDDEN\_SIZE. #则inputs的维度为：batch\_size \* num\_steps \* HIDDEN\_SIZE 56. embedding**=**tf.get\_variable('embedding',[VOCAB\_SIZE,HIDDEN\_SIZE]) 57. inputs**=**tf.nn.embedding\_lookup(embedding,self.input\_data) 59. # 如果在训练阶段，则进行dropout 60. **if** is\_training: 61. inputs**=**tf.nn.dropout(inputs,KEEP\_PROB) 63. #定义LSTM结构的输出列表 64. outputs**=**[] 65. # state存储LSTM的初始状态 66. state**=**self.initial\_state 67. #TensorFlow提供了Variable Scope 机制，用于共享变量 68. with tf.variable\_scope('RNN'): 69. #对于每个时间步 70. **for** time\_step **in** range(num\_steps): 71. **if** time\_step>0: 72. tf.get\_variable\_scope().reuse\_variables() #重用变量 74. # 从输入数据inputs中获取当前时刻的输入并传入LSTM的单元，并得到输出。 75. #每次的输出都是一个张量，其shape=(20, 200)，其中20是BATCH\_SIZE，200是词向量维度 76. cell\_output,state**=**cell(inputs[:,time\_step,:],state) 77. #将当前单元的输出加入到输出的列表中 78. outputs.append(cell\_output) 80. # 将输出的列表利用tf.concat函数变成（batch,hidden\_size\*num\_steps）的形状，然后再reshape成（batch\*num\_steps,hidden\_size）的形状 81. #,即为 （20\*35， 200）=（700, 200） 82. output **=** tf.reshape(tf.concat(outputs, 1), [**-**1, HIDDEN\_SIZE]) 84. #下面三条语句利用刚才LSTM的输出向量output乘以权重weight再加上偏置bias，得到最后的预测结果logits 85. weight**=**tf.get\_variable('weight',[HIDDEN\_SIZE,VOCAB\_SIZE])  #weight的形状为 [200, 10000] 86. bias**=**tf.get\_variable('bias',[VOCAB\_SIZE]) 87. logits**=**tf.matmul(output,weight)**+**bias 89. #计算交叉熵损失 90. loss**=**tf.contrib.legacy\_seq2seq.sequence\_loss\_by\_example( 91. [logits], #logits是刚才预测的结果  shape=(700, 10000) 92. [tf.reshape(self.targets,[**-**1])], # self.targets是正确的结果.这里对其shape进行了调整，变为shape=(700,) 93. [tf.ones([batch\_size**\***num\_steps],dtype**=**tf.float32)]# 设置损失的权重。所有的权重都设为1，即不同batch和不同时刻的权重相同 94. ) 96. # 计算每个batch的平均损失，并更新网络的状态 97. self.cost**=**tf.reduce\_sum(loss)**/**batch\_size 98. self.final\_state**=**state 100. # 如果当前是在训练阶段，则继续进行下面的反向传播以更新梯度 101. **if** **not** is\_training: 102. **return** 103. #返回的是需要训练的变量列表 104. trainable\_variables**=**tf.trainable\_variables() 106. # 通过clip\_by\_global\_norm函数控制梯度的大小，避免梯度膨胀问题 107. grads,\_**=**tf.clip\_by\_global\_norm( 108. tf.gradients(self.cost,trainable\_variables),MAX\_GRAD\_NORM) 110. #定义模型的优化方法 111. optimizer**=**tf.train.GradientDescentOptimizer(LEARNING\_RATE) 112. #应用梯度对trainable\_variables进行更新 113. self.train\_op**=**optimizer.apply\_gradients(zip(grads,trainable\_variables))  116. # 定义的函数run\_epoch。使用刚才定义的模型model在数据data上运行train\_op并返回perplexity值 117. **def** run\_epoch(session,model,data,train\_op,output\_log,epoch\_size): 118. # 这两个变量是用于计算perplexity的辅助中间变量 119. total\_costs**=**0.0 120. iters**=**0 121. #对模型进行初始化操作 122. state**=**session.run(model.initial\_state) 123. # 对每一轮，使用当前数据训练模型 124. **for** step **in** range(epoch\_size): 126. #这个run语句会返回并得到预测值y 127. x, y **=** session.run(data) 129. #下面的run语句计算得到cost，即交叉熵值 130. cost,state,\_**=**session.run([model.cost, model.final\_state, train\_op], 131. {model.input\_data:x, model.targets:y, model.initial\_state:state}) 133. #【然后求和】 134. total\_costs**+=**cost 135. iters**+=**model.num\_steps 137. # 如果output\_log为真，且每隔100步，就会输出困惑度的数值。困惑度ppx=exp(loss/N) 138. **if** output\_log **and** step **%** 100 **==**0: 139. print('After {} steps,perplexity is {}'.format(step,np.exp(total\_costs**/**iters))) 141. # 返回perplexity值 142. **return** np.exp(total\_costs**/**iters) 144. #主函数的定义 145. **def** main(\_): 146. # 利用reader.ptb\_raw\_data 函数 从数据目录DATA\_PATH中读取训练数据集，验证数据集和测试数据集 147. train\_data,valid\_data,test\_data,\_**=**reader.ptb\_raw\_data(DATA\_PATH) 149. #得到训练阶段中每个epoch所需要训练的次数，即 train\_epoch\_size。 150. train\_data\_len **=** len(train\_data)  #得到训练数据的文本长度 151. train\_batch\_len **=** train\_data\_len **//** TRAIN\_BATCH\_SIZE  #得到训练数据共计需要多少个batch 152. train\_epoch\_size **=** (train\_batch\_len **-** 1) **//** TRAIN\_NUM\_STEP  #再除以时间步的个数，从而得到每个epoch所需要训练的次数 154. #下同 155. valid\_data\_len **=** len(valid\_data) 156. valid\_batch\_len **=** valid\_data\_len **//** EVAL\_BATCH\_SIZE 157. valid\_epoch\_size **=** (valid\_batch\_len **-** 1) **//** EVAL\_NUM\_STEP 159. test\_data\_len **=** len(test\_data) 160. test\_batch\_len **=** test\_data\_len **//** EVAL\_BATCH\_SIZE 161. test\_epoch\_size **=** (test\_batch\_len **-** 1) **//** EVAL\_NUM\_STEP 163. #利用tf.random\_uniform\_initializer生成均匀分布的随机数 164. initializer**=**tf.random\_uniform\_initializer(**-**0.05,0.05) 166. #定义训练阶段使用的模型，PTBModel是之前定义过的类，其定义了网络结构。 167. with tf.variable\_scope('language\_model',reuse**=**None,initializer**=**initializer): 168. train\_model**=**PTBModel(True,TRAIN\_BATCH\_SIZE,TRAIN\_NUM\_STEP) 170. #定义测试阶段使用的模型 171. with tf.variable\_scope('language\_model',reuse**=**True,initializer**=**initializer): 172. eval\_model**=**PTBModel(False,EVAL\_BATCH\_SIZE,EVAL\_NUM\_STEP) 174. #使用with tf.Session()语句来创建上下文（Context）并执行 175. with tf.Session() as sess: 176. #初始化变量 177. tf.global\_variables\_initializer().run() 179. #利用reader.ptb\_producer函数切分生成训练集、验证集和测试集上各自的队列 181. train\_queue **=** reader.ptb\_producer(train\_data, train\_model.batch\_size, train\_model.num\_steps) 182. eval\_queue **=** reader.ptb\_producer(valid\_data, eval\_model.batch\_size, eval\_model.num\_steps) 183. test\_queue **=** reader.ptb\_producer(test\_data, eval\_model.batch\_size, eval\_model.num\_steps) 185. #使用tf.train.Coordinator()来创建一个线程管理器对象。然后调用tf.train.start\_queue\_runners，把数据推入内存序列中供计算单元调用 186. coord **=** tf.train.Coordinator() 187. threads **=** tf.train.start\_queue\_runners(sess**=**sess, coord**=**coord) 189. # 对于每个epoch进行训练。NUM\_EPOCH表示总的训练的轮数 190. **for** i **in** range(NUM\_EPOCH): 191. print('In EPOCH: {}'.format(i**+**1)) #【对于每个epoch】 192. # 调用之前的run\_epoch函数来训练模型，当然要提供训练的模型，训练数据等 193. run\_epoch(sess,train\_model,train\_queue,train\_model.train\_op,True,train\_epoch\_size) 195. #使用验证数据来计算模型的困惑度，并打印输出 196. valid\_perplexity**=**run\_epoch(sess,eval\_model,eval\_queue,tf.no\_op(),False,valid\_epoch\_size) 197. print('Epoch: {} Validation Perplexity: {}'.format(i**+**1,valid\_perplexity)) 199. # 计算并返回在测试数据集上的困惑度，然后打印输出 200. test\_perplexity**=**run\_epoch(sess,eval\_model,test\_queue,tf.no\_op(),False,test\_epoch\_size) 201. print('Test Perplexity: {}'.format(test\_perplexity))  204. #使用coord.request\_stop()来发出终止所有线程的命令，使用coord.join(threads)把线程加入主线程，等待threads结束。 205. coord.request\_stop() 206. coord.join(threads) 208. #主函数的入口 210. **if** \_\_name\_\_ **==** '\_\_main\_\_': 211. tf.app.run() |

1. **卷积神经网络**

**6.1 卷积与卷积神经网络(CNN)**

* **什么是卷积(Convolution)?**

|  |
| --- |
|  |

* **什么是卷积神经网络**

|  |
| --- |
|  |

|  |
| --- |
|  |

* **CNN特点之一：局部卷积**

|  |
| --- |
|  |

* **CNN特点之二：参数共享**

|  |
| --- |
|  |

* **CNN特点之三：多卷积核**

|  |
| --- |
|  |

* **CNN特点之四：池化处理**

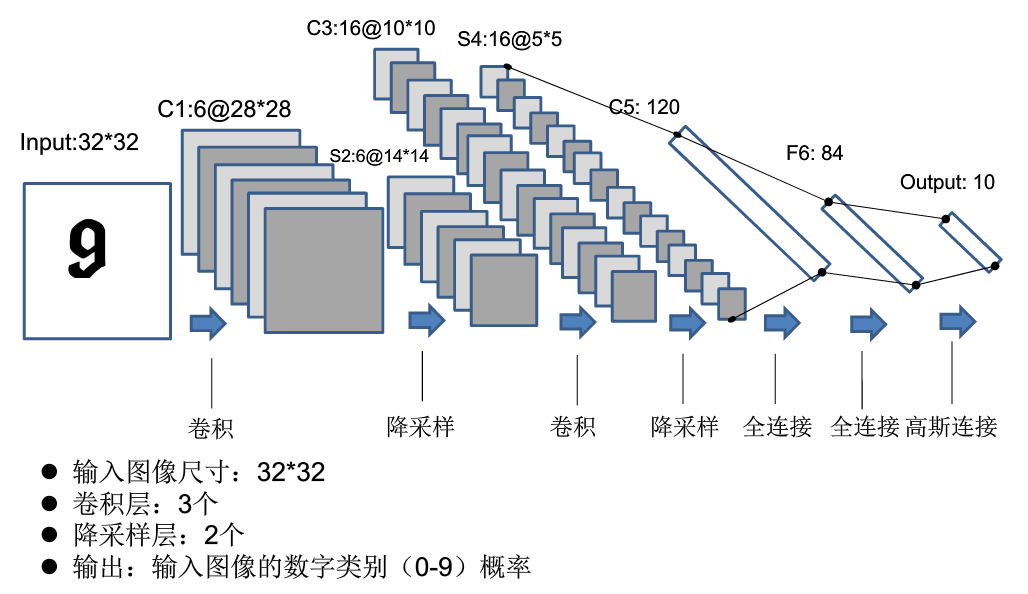
|  |
| --- |
|  |

* **CNN特点之五：池化处理**

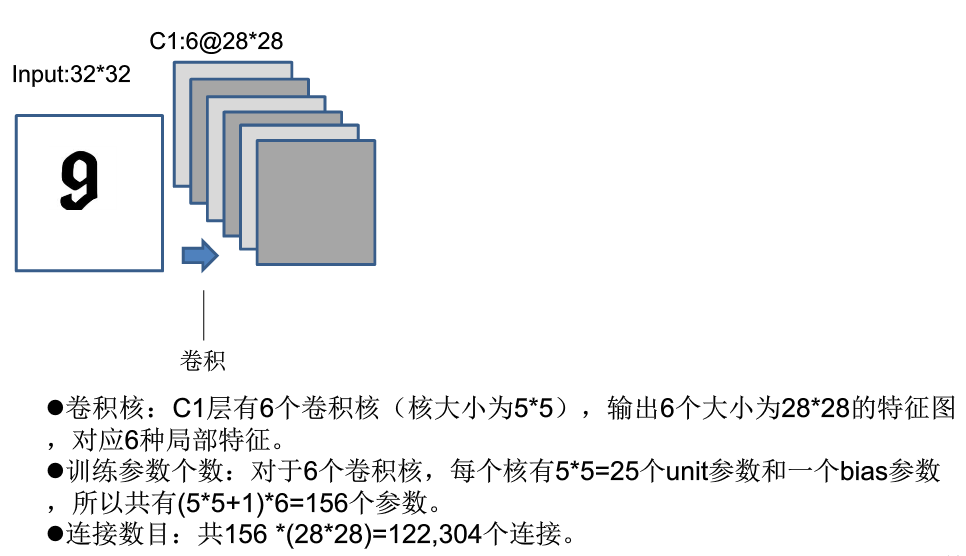
|  |
| --- |
|  |

**6.2 LeNet-5 模型分析**

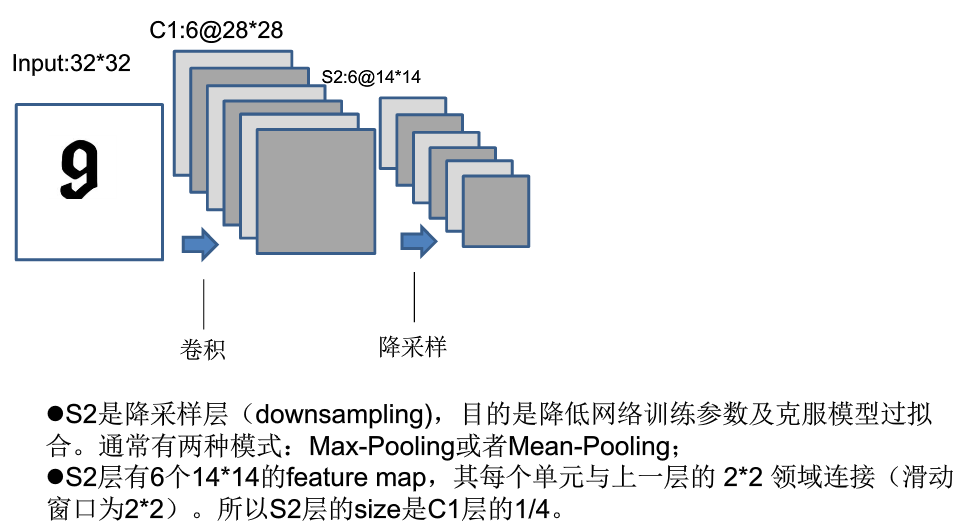
* **LeNet-5**



* **LeNet-5——卷积层C1**



* **LeNet-5——降采样层S2**



* **LeNet-5——卷积层C3**

|  |
| --- |
|  |

* **LeNet-5——降采样层S4**

|  |
| --- |
|  |

* **LeNet-5——卷积层C5**

|  |
| --- |
|  |

* **LeNet-5的全连接**

|  |
| --- |
|  |

* **LeNet-5的输出层**

|  |
| --- |
|  |

* **基于CNN的文本分类**

|  |
| --- |
|  |

* **CNN代码学习笔记**

|  |
| --- |
|  |

**6.3 卷积神经网络的程序示例**

|  |
| --- |
| 1. #演示内容：利用tensorflow实现的卷积神经网络来进行MNIST手写数字图像的分类。 3. #导入numpy模块 4. **import** numpy as np 5. #导入tensorflow模块，程序使用tensorflow来实现卷积神经网络 6. **import** tensorflow as tf 8. #下载mnist数据集，并从mnist\_data目录中读取数据 9. **from** tensorflow.examples.tutorials.mnist **import** input\_data 10. mnist **=** input\_data.read\_data\_sets('mnist\_data',one\_hot**=**True) 11. #（1）这里的“mnist\_data” 是和当前文件相同目录下的一个文件夹。自己先手工建立这个文件夹，然后从https://yann.lecun.com/exdb/mnist/ 下载所需的4个文件（即该网址中第三段“Four files are available on this site:”后面的四个文件），并放到目录MNIST\_data下即可。  #（2）MNIST数据集是手写数字字符的数据集。每个样本都是一张28\*28像素的灰度手写数字图片。 12. #（3）one\_hot表示独热编码，其值被设为true。在分类问题的数据集标注时，如何不采用独热编码的方式， 类别通常就是一个符号而已，比如说是9。但如果采用独热编码的方式，则每个类表示为一个列表list，共计有10个数值，但只有一个为1，其余均为0。例如，“9”的独热编码可以为[00000 00001]. 14. #定义输入数据x和输出y的形状。函数tf.placeholder的目的是定义输入，可以理解为采用占位符进行占位。 15. #None这个位置的参数在这里被用于表示样本的个数，而由于样本个数此时具体是多少还无法确定，所以这设为None。而每个输入样本的特征数目是确定的，即为28\*28。 16. input\_x **=** tf.placeholder(tf.float32,[None,28**\***28])**/**255 #因为每个像素的取值范围是 0～255 17. output\_y **=** tf.placeholder(tf.int32,[None,10])  #10表示10个类别 19. #输入层的输入数据input\_x被reshape成四维数据，其中第一维的数据代表了图片数量 20. input\_x\_images **=** tf.reshape(input\_x,[**-**1,28,28,1]) 21. test\_x **=** mnist.test.images[:3000] #读取测试集图片的特征，读取3000个图片 22. test\_y **=** mnist.test.labels[:3000] #读取测试集图片的标签。就是这3000个图片所对应的标签 24. #定义卷积神经网络CNN的结构 25. #首先定义第一层，即卷积层conv1的结构。tf.layers.conv2d函数表示要使用2维的卷积层 26. conv1 **=** tf.layers.conv2d( 27. inputs**=**input\_x\_images,#输入是input\_x\_images 。 28. filters**=**32,#滤波器的数量是32。 29. kernel\_size**=**[5,5],#卷积核的尺寸即kernel\_size是5\*5。 30. strides**=**1,#strides即步长设为1. 31. padding**=**'same',#padding参数的作用是决定在进行卷积时，是否要 对输入的图像矩阵边缘补0。 32. activation**=**tf.nn.relu#最后设置activation，即激活函数设为relu。 33. ) 34. print(conv1) #打印conv1，其输出的shape为[28,28，32] 36. #利用tf.layers.max\_pooling2d 函数进行最大池操作，此时的输入就是刚才的conv1，这里pool\_size设为2\*2，步长为2。 37. pool1 **=** tf.layers.max\_pooling2d( 38. inputs**=**conv1, 39. pool\_size**=**[2,2], 40. strides**=**2 41. ) 42. print(pool1) #用print语句，会看到其输出的形状为[14,14,32]，由于每个图像变为了14\*14的，因此图片的尺寸明显变小了 44. #定义了第二次卷积的结构。同样使用tf.layers.conv2d函数。输入就是刚才的最大池处理后的结果。这里使用了64个卷积核，每个核的大小为5\*5. 其它的设置和第一个卷积层相同。 45. conv2 **=** tf.layers.conv2d( 46. inputs**=**pool1, 47. filters**=**64, 48. kernel\_size**=**[5,5], 49. strides**=**1, 50. padding**=**'same', 51. activation**=**tf.nn.relu 52. ) 53. print(conv2) #由于padding设为了'same'，所以输出为[14,14,64]。所以这一层卷积处理后，特征图的数目增加了。但每个图的尺寸仍然是14\*14 55. #利用tf.layers.max\_pooling2d 函数进行再一次的最大池操作。此时的输入就是刚才的conv2，pool\_size设为2\*2，步长为2。 56. pool2 **=** tf.layers.max\_pooling2d( 57. inputs**=**conv2, 58. pool\_size**=**[2,2], 59. strides**=**2 60. ) 61. print(pool2) #输出为[7,7,64]，即尺寸发生了变化，变为7\*7 63. #刚才进行的是两次卷积以及降采样。得到的特征图有64个，每个大小为7\*7=49。下面就要对每个图进行打平处理，64\*49=3136. 这样做的目的是为了后续的全连接层做准备，以便得到softmax数值】 64. flat **=** tf.reshape(pool2,[**-**1,7**\***7**\***64]) 66. #下面接一个全连接层，使用的是tf.layers.dense函数，其输入为刚才打平的结果flat。units: 输出的大小（维数），激活函数设为relu 67. dense**=**tf.layers.dense( 68. inputs**=**flat, 69. units**=**1024, 70. activation**=**tf.nn.relu 71. ) 72. print(dense) #输出的特征图的像素个数应该是1024 74. #利用tf.layers.dropout函数做了一次dropout操作。丢弃率rate设置为0.5，即一半的神经元丢弃不工作。输入就是dense 75. dropout **=** tf.layers.dropout( 76. inputs**=**dense, 77. rate**=**0.5 78. ) 79. print(dropout) #输出应该仍为1024个数值 81. #使用tf.layers.dense定义输出层，输入是刚才的dropout的结果。这个层是一个简单的全连接层，没有使用激活函数，输出10个数值。 82. outputs **=** tf.layers.dense( 83. inputs**=**dropout, 84. units**=**10 85. ) 86. print(outputs) #输出为10个数值 88. #到现在为止，完成了一次正向传播。此时得到的outputs就是预测的结果。】 89. #计算交叉熵损失，利用的是softmax\_cross\_entropy函数。其中的output\_y之前讲过，表示真实的分类结果。而outputs则表示预测的结果 90. loss **=** tf.losses.softmax\_cross\_entropy(onehot\_labels**=**output\_y,logits**=**outputs) 92. print(loss)#打印这个loss的维度 94. #定义训练操作，采用梯度下降的优化方法，设置学习率为0.001 95. train\_op **=** tf.train.GradientDescentOptimizer(0.001).minimize(loss) 97. #定义模型的性能评价指标为准确率，具体是使用tf.metrics.accuracy函数来实现的。labels是正确的标签，predictions则是预测的结果 98. accuracy\_op **=** tf.metrics.accuracy( 99. labels**=**tf.argmax(output\_y,axis**=**1), #返回张量维度上最大值的索引 100. predictions**=**tf.argmax(outputs,axis**=**1)#返回张量维度上最大值的索引 101. ) 103. print(accuracy\_op)#打印这个张量 105. #以上定义了tensorflow的图，下面就要开始执行任务了。首先要初始化所有变量 106. sess**=**tf.Session() 107. init**=**tf.group(tf.global\_variables\_initializer(),tf.local\_variables\_initializer()) 108. sess.run(init) 109. #只训练了1000个批次，即进行了1000次迭代 110. **for** i **in** range(1000): 111. #每次迭代只选取 50个图片及其标签】 112. batch **=** mnist.train.next\_batch(50) 113. #把batch中的这50幅图像的特征数据batch[0]及其标签batch[1] 送入模型，进行训练操作，并得到其训练损失 114. train\_loss, train\_op\_**=**sess.run([loss,train\_op],{input\_x:batch[0],output\_y:batch[1]}) 115. #每迭代100次就输出一下训练的损失函数取值和在测试数据集（只选了3000个图片）上的准确率。 116. **if** i**%**100**==**0: 117. #计算在测试集上的准确率 118. test\_accuracy**=**sess.run(accuracy\_op,{input\_x:test\_x,output\_y:test\_y}) 119. print("Step=%d, Train loss=%.4f,Test accuracy=%.2f"**%**(i,train\_loss,test\_accuracy[0]))#打印输出当前的步数，训练的损失，以及准确率 121. #这个模型已经训练完毕。下面选取20幅图像，看一下模型预测的效果如何。这里test\_output就是预测的结果 122. test\_output**=**sess.run(outputs,{input\_x:test\_x[:20]}) 123. #由于test\_output实际上是一个softmax的结果，所以下面用np.argmax函数求每一幅图像预测的分类值到底是几 124. inferenced\_y**=**np.argmax(test\_output,1) 125. #所以这个inferenced\_y，实际上就是预测的数字到底是几，把它打印出来。 126. print(inferenced\_y,'Inferenced numbers（预测的标签）') 127. #下面我们再使用print语句，看一下这20个图像的真实标签 128. print(np.argmax(test\_y[:20],1),'Real numbers（真实的标签') 129. #关闭这个session 130. sess.close() |

1. **递归神经网络**

**7.1 情感分析（Sentiment Analysis）**

* **什么是情感分析**

|  |
| --- |
|  |

* **情感分类和文本分类的关系**

|  |
| --- |
|  |

**7.2 词向量**

* **深度学习中的词向量表示**

|  |
| --- |
|  |

* **利用Word2vec得到词向量**

|  |
| --- |
|  |

* **利用Word2vec进行词的相似度计算**

|  |
| --- |
|  |

* **其他词向量表示工具**

|  |
| --- |
|  |

**7.3 递归神经网络及其变体（Recursive Neural Networks）**

* **如何判断一段文本的情感倾向？**

|  |
| --- |
|  |

* **情感树库语料**

|  |
| --- |
|  |
|  |

* **标准RNN**

|  |
| --- |
|  |

* **RNTN**

|  |
| --- |
|  |

1. **生成式神经网络**

**8.1 自动编码器**

* **有监督学习**

|  |
| --- |
|  |

* **无监督学习**

|  |
| --- |
|  |

* **生成式模型（Generative Models）**
* 从训练数据中学习，其与训练数据中的分布尽可能接近；
* 然后中生成新的样本

|  |
| --- |
|  |

* **自动编码器（Autoencoder）**

|  |
| --- |
|  |

* **只有一个隐藏层的自动编码器**

|  |
| --- |
|  |

* **自动编码器的损失函数**

|  |
| --- |
|  |

* **堆栈自动编码器**

|  |
| --- |
|  |

* **用自动编码器得到新的特征**

|  |
| --- |
|  |

**8.2 变分自动编码器(Variational Autoencoders)**

* **变分自动编码器-图像生成**

|  |
| --- |
|  |

* **变分自动编码器的原理**

|  |
| --- |
|  |

* **变分自动编码器中的分布采样**

|  |
| --- |
|  |

* **变分自动编码器中的对数似然函数**

|  |
| --- |
|  |

* **变分自动编码器中的训练过程**

|  |
| --- |
|  |

* **生成新的样本**

|  |
| --- |
|  |

**8.3 生成对抗网络**

* **判别式模型和生成式模型**

|  |
| --- |
|  |

* **生成对抗网络GAN的基本原理**

|  |
| --- |
|  |

* **判断模型的目标函数**

|  |
| --- |
|  |

* **生成模型的目标函数**

|  |
| --- |
|  |

* **GAN训练算法**

|  |
| --- |
|  |

* **GAN训练中的参数更新**

|  |
| --- |
|  |

* **GAN在MNIST上的运行效果**

|  |
| --- |
|  |

**8.4 自动编码器程序示例**

|  |
| --- |
| 1. #演示内容：利用自动编码器实现手写数字图像数据集的二维平面的展示 3. **import** tensorflow as tf 4. **import** matplotlib.pyplot as plt 6. **from** tensorflow.examples.tutorials.mnist **import** input\_data 7. mnist **=** input\_data.read\_data\_sets("MNIST\_data/", one\_hot**=**False)#独热编码设为False 9. # xs, ys = mnist.train.next\_batch(10)  #这个函数是取出一些数据。是从训练集里一次提取10张图片。则xs中存储的是图像的数据，ys存储的是这些图像对应的类别 10. # print (ys)  #此时打印的应该是类别 11. # mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True) 12. # xs,ys = mnist.train.next\_batch(10) 13. # print (ys)  #此时打印的应该是独热编码的形式  16. learning\_rate **=** 0.01 17. training\_epochs **=** 10 18. batch\_size **=** 256 #每次迭代所使用的样本的数目； 19. display\_step **=** 1 20. n\_input **=** 784  #每个样本都是一张28 \* 28像素的灰度手写数字图片。 22. #定义输入，可以理解为采用占位符进行占位，待实际运行时进行具体填充,None表示运行时指定,是指样本的个数。 23. X **=** tf.placeholder("float", [None, n\_input]) 25. n\_hidden\_1 **=** 128 26. n\_hidden\_2 **=** 64 27. n\_hidden\_3 **=** 10 28. n\_hidden\_4 **=** 2 29. weights **=** { 30. 'encoder\_h1': tf.Variable(tf.truncated\_normal([n\_input, n\_hidden\_1],)), 31. 'encoder\_h2': tf.Variable(tf.truncated\_normal([n\_hidden\_1, n\_hidden\_2],)), 32. 'encoder\_h3': tf.Variable(tf.truncated\_normal([n\_hidden\_2, n\_hidden\_3],)), 33. 'encoder\_h4': tf.Variable(tf.truncated\_normal([n\_hidden\_3, n\_hidden\_4],)), 34. 'decoder\_h1': tf.Variable(tf.truncated\_normal([n\_hidden\_4, n\_hidden\_3],)), 35. 'decoder\_h2': tf.Variable(tf.truncated\_normal([n\_hidden\_3, n\_hidden\_2],)), 36. 'decoder\_h3': tf.Variable(tf.truncated\_normal([n\_hidden\_2, n\_hidden\_1],)), 37. 'decoder\_h4': tf.Variable(tf.truncated\_normal([n\_hidden\_1, n\_input],)), 38. } 39. biases **=** { 40. 'encoder\_b1': tf.Variable(tf.random\_normal([n\_hidden\_1])), 41. 'encoder\_b2': tf.Variable(tf.random\_normal([n\_hidden\_2])), 42. 'encoder\_b3': tf.Variable(tf.random\_normal([n\_hidden\_3])), 43. 'encoder\_b4': tf.Variable(tf.random\_normal([n\_hidden\_4])), 44. 'decoder\_b1': tf.Variable(tf.random\_normal([n\_hidden\_3])), 45. 'decoder\_b2': tf.Variable(tf.random\_normal([n\_hidden\_2])), 46. 'decoder\_b3': tf.Variable(tf.random\_normal([n\_hidden\_1])), 47. 'decoder\_b4': tf.Variable(tf.random\_normal([n\_input])), 48. } 49. **def** encoder(x): 50. layer\_1 **=** tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder\_h1']), 51. biases['encoder\_b1'])) 52. layer\_2 **=** tf.nn.sigmoid(tf.add(tf.matmul(layer\_1, weights['encoder\_h2']), 53. biases['encoder\_b2'])) 54. layer\_3 **=** tf.nn.sigmoid(tf.add(tf.matmul(layer\_2, weights['encoder\_h3']), 55. biases['encoder\_b3'])) 56. # 为了便于编码层的输出，编码层随后一层不使用激活函数 57. layer\_4 **=** tf.add(tf.matmul(layer\_3, weights['encoder\_h4']), 58. biases['encoder\_b4']) 59. **return** layer\_4 61. **def** decoder(x): 62. layer\_1 **=** tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder\_h1']), 63. biases['decoder\_b1'])) 64. layer\_2 **=** tf.nn.sigmoid(tf.add(tf.matmul(layer\_1, weights['decoder\_h2']), 65. biases['decoder\_b2'])) 66. layer\_3 **=** tf.nn.sigmoid(tf.add(tf.matmul(layer\_2, weights['decoder\_h3']), 67. biases['decoder\_b3'])) 68. layer\_4 **=** tf.nn.sigmoid(tf.add(tf.matmul(layer\_3, weights['decoder\_h4']), 69. biases['decoder\_b4'])) 70. **return** layer\_4 72. encoder\_op **=** encoder(X) 73. decoder\_op **=** decoder(encoder\_op) 75. y\_pred **=** decoder\_op 76. y\_true **=** X 78. cost **=** tf.reduce\_mean(tf.pow(y\_true **-** y\_pred, 2)) 79. optimizer **=** tf.train.AdamOptimizer(learning\_rate).minimize(cost) 81. with tf.Session() as sess: 82. # tf.initialize\_all\_variables() no long valid from 83. # 2017-03-02 if using tensorflow >= 0.12 84. **if** int((tf.\_\_version\_\_).split('.')[1]) < 12 **and** int((tf.\_\_version\_\_).split('.')[0]) < 1: 85. init **=** tf.initialize\_all\_variables() 86. **else**: 87. init **=** tf.global\_variables\_initializer() 88. sess.run(init) 89. total\_batch **=** int(mnist.train.num\_examples**/**batch\_size) 90. **for** epoch **in** range(training\_epochs): 91. **for** i **in** range(total\_batch): 92. batch\_xs, batch\_ys **=** mnist.train.next\_batch(batch\_size) 93. \_, c **=** sess.run([optimizer, cost], feed\_dict**=**{X: batch\_xs}) 94. **if** epoch **%** display\_step **==** 0: 95. print("Epoch:", '%04d' **%** (epoch**+**1), "cost=", "{:.9f}".format(c)) 96. print("Optimization Finished!") 98. encoder\_result **=** sess.run(encoder\_op, feed\_dict**=**{X: mnist.test.images}) 99. plt.scatter(encoder\_result[:, 0], encoder\_result[:, 1], c**=**mnist.test.labels) 100. plt.colorbar() 101. plt.show() |