卷积神经网络调研报告

PB16030899 朱河勤

Table of Content

---

1. 背景

卷积神经网络受生物的启发, 最早出现于 Hubel, Wiesel 关于猫的视觉神经网络[1]. 生物学中, 我们知道视觉神经网络包含很多神经细胞, 形成一个复杂, 庞大的排序. 这些细胞对小范围的视觉变化很敏感, 就像一个滤波器一样将看到的图片的视觉信息转化成一些稀疏的, 相关联的一些特征.

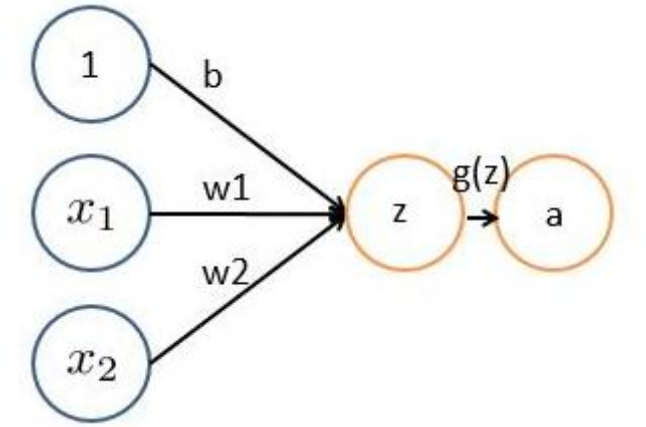
卷积神经网络就是模拟生物上的神经网络, 主要用于图片处理, 音频处理上, 是深度学习的一个重要工具.

2. 原理

下面根据卷积神经网络的结构介绍, 卷积网络如神经网络一样, 由由大量的神经元连接而成, 可以进行分层, 如同流水线一样, 每层的工作不同, 上一层的输出是下一层的输入.

2.1. 神经元

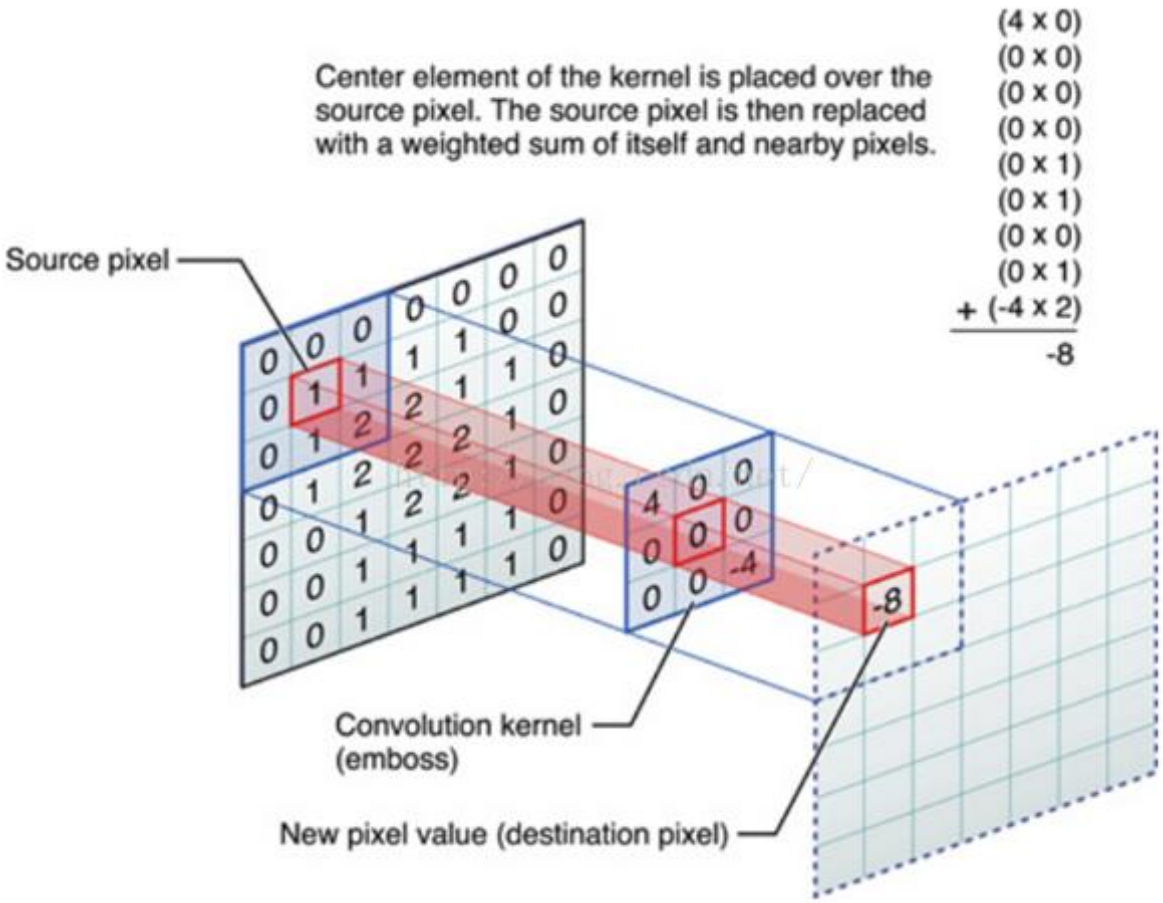
对于每个神经元,结构如下,



- x_1, x_2表示 输入, 一般形式是一个向量 X
- w_1, w_2表示权重, 一般情况下是W
- b为偏置 bias
- g(z)为激活函数
- a 为输出
  一般形式就是 $a = g(z) = W \cdot X + b$
  即X 向量表示输入信息, 每个维度对于输出的贡献不同, 于是加权 w, 然后通过 偏置 bias 设置相对值, 最后通过 激活函数将输出域限定在 $[0, 1]$, 这样可以表示概率[2], 也能归一化.

2.2. 卷积层(convolution)

对于一个输入矩阵, 如图形的像素信息, 这一层的神经元, 如果一个滤波器, 将输入中的局部信息进行卷积处理. 这里就是计算向量内积



滤波器 filter（带着一组固定权重的神经元）对局部输入数据进行卷积计算。每计算完一个数据窗口内的局部数据后，数据窗口不断平移滑动，直到计算完所有数据。这个过程中，有这么几个参数:

1. 深度 depth：神经元个数，决定输出的 depth 厚度。同时代表滤波器个数。
2. 步长 stride：决定滑动多少步可以到边缘
3. 填充值 zero-padding：在外围边缘补充若干圈 0，方便从初始位置以步长为单位可以刚好滑倒末尾位置，通俗地讲就是为了总长能被步长整除。

一个形象的比喻[2:1]是, 滤波器就像一双眼睛.

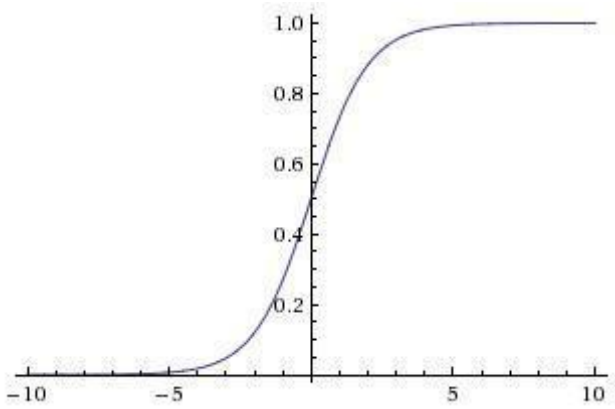
> 人类视角有限，一眼望去，只能看到这世界的局部。如果一眼就看到全世界，你会累死，而且一下子接受全世界所有信息，你大脑接收不过来。当然，即便是看局部，针对局部里的信息人类双眼也是有偏重、偏好的。比如看美女，对脸、胸、腿是重点关注，所以这 3 个输入的权重相对较大

有多少个 滤波器 就有多少个输出, 即有多少只眼睛去观察, 得到的信息也不同

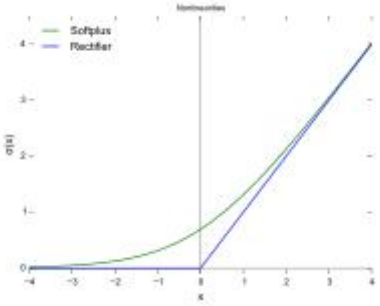
2.3. 激活层(activation)

这一层将输入数据经过激活函数处理, 归一化
激活函数一般是非线性的, 如

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



- ReLU[3]: $f(x) = x^+ = max(0, x)$, 这个函数来自生物学的事实, 是一个稀疏的激活函数, 由于简单, 很方便求梯度. 缺点是无界的, 而且在0处不可导.



选取好的函数对于神经网络的效果有很大的影响.
还有很多激活函数:

- Leaky ReLU
- Parametric ReLU
- Randomized ReLU
- ELU
- Maxout
- Probout

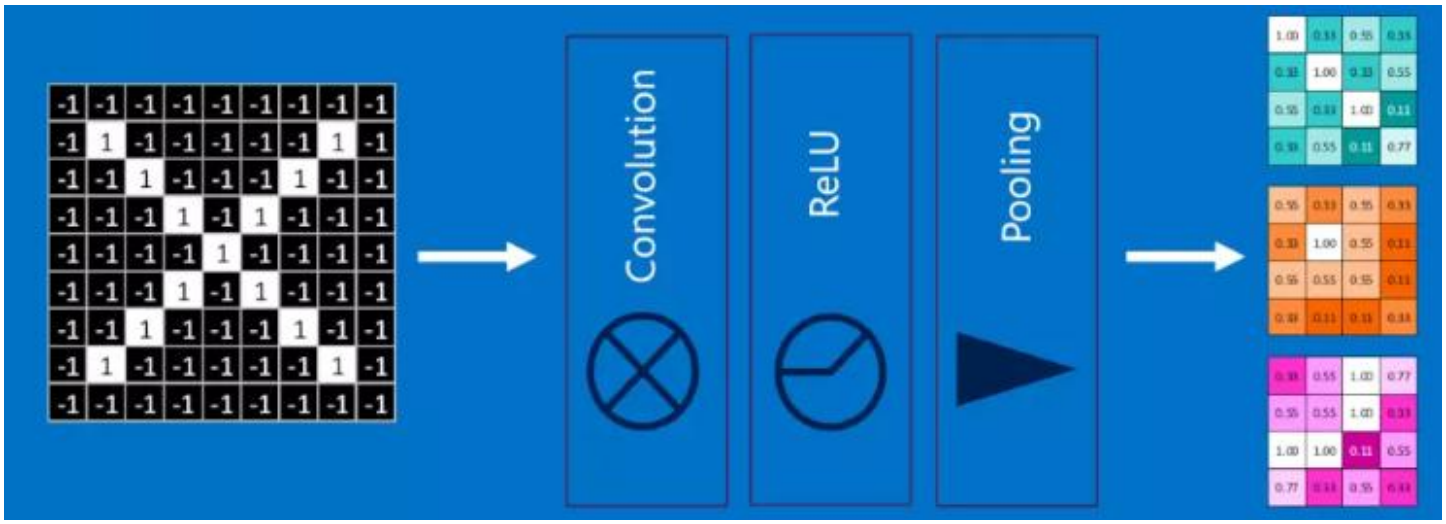
2.4. 池化层(pooling)

池化层的作用是减小输入规模, 同时保留重要看信息, 做法同样是像卷积层那样, 移动数据窗口, 对于每个窗口的信息, 缩小规模, 映射到一维的数, 可以是这个窗口的最大值 (这样保留了最佳匹配结果, 它只关注这个窗口是否具有某种特征, 而不用在意特征在哪里)[4], 也有其他方法, 如求这个窗口的平均值.

通过加入池化层, 可以很大程度上减少计算两, 降低机器负载

2.5. CNN

经过一系列流水线般的处理, 一层层网络的数据处理与传递, 就是下面这样



当然可以处理多层, 增加更多层的神经网络,如下

这些神经网络的参数通过 `BP(back propagation)` 训练, 即将输出结果再反向传播给输入, 直至 Error 值降到最低.

3. 应用

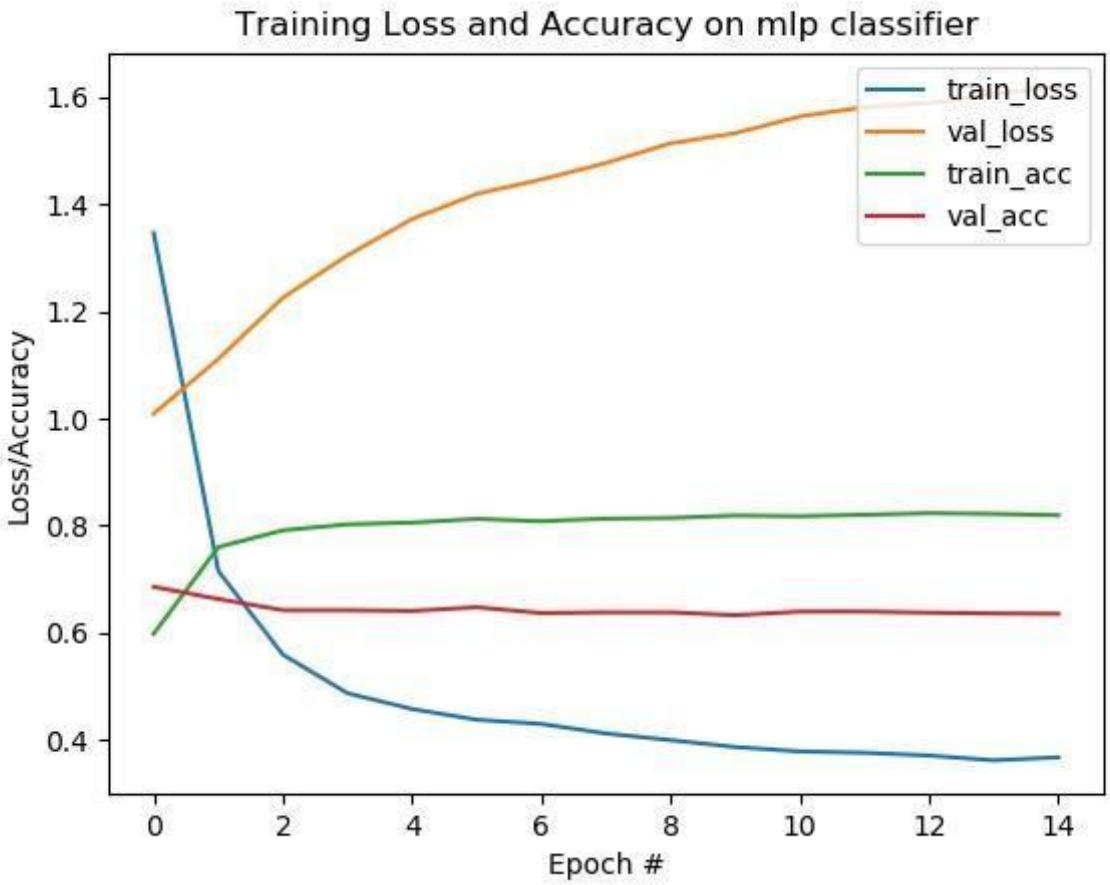
CNN 作为深度学习的主力, 有很多应用

- 图像分类
- 物体检测
- 物体追踪
- 姿态预估
- 文本检测识别
- 场景识别
  但最合适的还是图像方面. 需要注意的是 CNN 是提取的局部特征, 如可以很轻易地提取出物体轮廓, 但是不能较好地把握整体特征.

4. 实验

> *实验代码见附录*, 使用的数据 是 2018 datafountain 汽车观点比赛的数据

我进行了多层感知机的实验来观察 神经网络的效果.
使用的是一些文本评论数据, 由此来进行分类,



这是训练过程中的 loss 与 acc 作图, 可以发现, 当 train_loss 降低过程, val_loss 还逐渐增大, 说明过拟合了, 重新调整参数, 最终得到训练结果 $F_1 = 0.61$.

这是我对神经网络的一次小尝试. 我熟悉了这个流程:

1. 数据分析: 观察数据的特点, 考虑特征的提取, 组合方法, 模型的选取
2. 数据预处理: (对于中文, 要进行分词) 去除停用词(这些是噪声, 可能干扰模型), 进行编码, 转换成向量
3. 划分训练集与测试集, 进行训练并预测
4. 评估预测结果

5. 心得

5.1. 选取超参数

- **滤波器的个数**[5]:
  注意每个滤波器的计算需要的时间要比 传统的 MLP 时间多很多, 所以一般选取的较少的滤波器.

如对于 第$l-1$层, 有K^{l-1}个特征矩阵, 而输入矩阵(图片的像素矩阵)是$M \times N$, 滤波器大小是$m \times n$, 那么一个滤波器需要的时间复杂度为

$$(M-m) \times (N-n) \times m \times n \times K^{l-1}$$

而 MLP 只需 K^{l-1}

- **滤波器大小**:
  找到正确的粒度来抽象提取处特征, 对于28×28大小的图片常用5×5大小的滤波器, 对于几百大小像素的图片, 则用12×12左右的滤波器
- **池化大小**: 一般是2×2大小的, 如果输入很大, 可以稍微增大

5.2. CNN为什么有效

- 从神经科学角度: 受生物视觉系统的启发
- 从正则化角度: 由于权值共享, 降低了模型参数数量, 控制了模型复杂度
- 从统计角度: 卷积抓住了问题的**局部相关性** 和**空间不变性**, 对于图像,语音领域的数据, 共同特点就是具有局部相关性, 如像素点是周围的像素点是有关联的. CNN 低层次的特征经过组合形成高层次的特征能保持空间不变性

6. 附录--实验代码

```
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.layers.merge import concatenate
from keras.models import Sequential, Model
from keras.layers import Dense, Embedding, Activation, merge, Input, Lambda, Reshape
from keras.layers import Convolution1D, Flatten, Dropout, MaxPool1D, GlobalAveragePooling1D
from keras.layers import LSTM, GRU, TimeDistributed, Bidirectional
from keras.utils.np_utils import to_categorical
from keras import initializers
from keras import backend as K
from keras.engine.topology import Layer

from sklearn.preprocessing import LabelEncoder as LE
from sklearn.linear_model import SGDClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cross_validation import train_test_split

from keras.models import load_model

from matplotlib import pyplot as plt

import pandas as pd
import numpy as np

from tagging import matchEmoPos,matchSubPos
from jieba import posseg,cut
import pickle
import os

STOPWORD = 'stop_word.txt'
TRAINFILE = 'train.csv'
TESTFILE = 'test_public.csv'

TESTSIZE=0.5

RETRAIN = True
epoch= 5

def getObjClassName(obj):
    try:
        return obj.__class__.__name__
    except:
        return type(obj).__name__

def stopWord(file =STOPWORD):
    with open(file,'r',encoding='utf8') as f :
        s = f.read()
        return s.split('\n')

punctuation = ' 。 、 ， 、 ； ： “”‘’ 「 」 『 』 （ ） 〔 〕 【 】 … —- ～ 《 》 〈 〉 ﹏ ' +'"#$%&\'()*+,-./:;<=>?@[\\]^_
def cutWordEmo(txt):
    stops = punctuation
    segs = cut(txt)
    words = [i for i in segs if i not in stops]
```

```
    #segs = posseg.cut(txt)
    #words=[word  for word,flag in segs if word not in stop_words and  matchEmoPos(flag)]
    return ' '.join(words)

stop_words = set(stopWord())

stop_words.union(punctuation)
stop_words.union(('不错','差','还','行','有点','不','差不多'))


def cutWord(txt):
    #segs = cut(txt)
    #words = [i for i in segs if i not in stop_words]
    segs = posseg.cut(txt)
    words=[word  for word,flag in segs if word not in stop_words and  matchSubPos(flag)]
    return ' '.join(words)


def tokenize(data,tokenizer=None):
    '''分词，构建单词-id词典'''
    if tokenizer is None:
        tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',lower=True,split='
        tokenizer.fit_on_texts(data)
    # 将每个词用词典中的数值代替
    seq = tokenizer.texts_to_sequences(data)

    # 序列模式
    #ret = pad_sequences(seq,maxlen=20)

    # one-hot
    ret = tokenizer.sequences_to_matrix(seq, mode='binary')
    return ret,tokenizer

def labelize(data,le=None):
    if le is None:
        le = LE()
        le.fit(data)
    vecs = le.transform(data)
    return  to_categorical(vecs,len(le.classes_)),le#reverse func: argmax(x, axis=None, out=Nor
def preprocess(file,target):
    cols = ['content',target]
    df = pd.read_csv(file,usecols=cols)
    df = df.fillna('')
    #df['content']+= df['sentiment_word']*3
    df['content'] = df['content'].apply(cutWord)
    return df


def Model_MLP(file,target='subject'):
    if not RETRAIN and os.path.exists('mlp.h5'):
        tk, le = pickle.load(open('token-le.pkl','rb'))
        return load_model('mlp.h5'),tk,le

    df = preprocess(file,target)
    df_extra = preprocess('extra0.csv',target)
```

```
    y ,le= labelize(df[target])
    y2 ,le =  labelize(df_extra[target],le)

    xtrain,xtest,ytrain,ytest = train_test_split(df['content'], y, test_size=TESTSIZE)

    x2 = df_extra['content'].values

    xtrain = np.concatenate((xtrain,x2),axis=0)# differ from xtrain +=x2
    xtrain,tokenizer = tokenize(xtrain)
    xtest , tokenizer =tokenize(xtest,tokenizer)

    ytrain = np.concatenate((ytrain,y2),axis=0)

    model = Sequential()
    # 全连接层  output space (units = 512)
    model.add(Dense(512, input_shape=(xtrain.shape[1],), activation='relu'))
    # DropOut层
    model.add(Dropout(0.5))
    # 全连接层+分类器
    model.add(Dense(len(le.classes_),activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    Hist = model.fit(xtrain, ytrain,
              epochs=epoch, # 迭代次数
              validation_data=(xtest, ytest))
    drawHistory(Hist.history,epoch)
    model.save('mlp.h5')
    pickle.dump((tokenizer,le),open('token-le.pkl','wb'))
    return (model,tokenizer,le)

def drawHistory(history,epochs):
    plt.figure()
    plt.plot(np.arange(0, epochs), history["loss"], label="train_loss")
    plt.plot(np.arange(0, epochs), history["val_loss"], label="val_loss")
    plt.plot(np.arange(0, epochs), history["acc"], label="train_acc")
    plt.plot(np.arange(0, epochs), history["val_acc"], label="val_acc")
    plt.title("Training Loss and Accuracy on mlp classifier")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss/Accuracy")
    plt.legend(loc="upper right")
    plt.savefig("Loss_Accuracy_mlp_{:d}e_extra.jpg".format(epochs))


def drawGroupCount(df,groupCol,countCol):
    fig = plt.figure(figsize=(8,6))
    getattr(df.groupby(groupCol),countCol).count().plot.bar(ylim=0)
    plt.show()


def Model_LSTM(file,target='subject',retrain=RETRAIN):
    if not retrain and os.path.exists('lstm.h5'):
        tk, le = pickle.load(open('token-le.pkl','rb'))
        return load_model('mlp.h5'),tk,le
```

```
    #numpy.ndarray
    x,y,tokenizer, le = preprocess(file,target)
    xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=TESTSIZE, random_state=42)

    model = Sequential()
    model.add(Embedding(xtrain.shape[1], 256))
    model.add(LSTM(256, 128)) # try using a GRU instead, for fun
    model.add(Dropout(0.5))
    model.add(Dense(128, 1))
    model.add(Activation('sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', class_mode="binary")
    model.fit(xtrain, ytrain, batch_size=16, nb_epoch=20)

def predict(model=Model_MLP):
    mdl, tk, le = model(TRAINFILE)

    df = pd.read_csv(TESTFILE,usecols=['content_id','content'],index_col=['content_id'])
    df['content'] = df['content'].apply(cutWord)
    X,tk = tokenize(df['content'], tk) # X is tuple, can't be used to predict
    df = df.drop('content',axis=1)
    Y = mdl.predict(X)
    y = [np.argmax(i, axis=None, out=None)  for i in Y]
    df['subject'] = le.inverse_transform(y)

    df['sentiment_value'] = 0
    df['sentiment_word'] = ''
    df.to_csv('submit-mlp.csv',index=True,sep=',',encoding='utf-8')

if __name__ =='__main__':
    predict()
```

7. 参考文献

1. Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (London), 195, 215–243. ↩
2. July. CNN 笔记：通俗理解卷积神经网络, 2016-7-2, csdn blog ↩ ↩
3. Rectifier (neural networks). wikipedia ↩
4. [透析] 卷积神经网络 CNN 究竟是怎样一步一步工作的? ↩
5. Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation. DeepLearning 0.1. LISA Lab. [31 August 2013]. ↩