**人工智能导论课程报告**

学院：信息学院 班级：理科试验班3班 姓名：范卓娅 学号：2016202160

大一刚入学的新生研讨课上，老师向我们介绍了微软小冰等人工智能领域的应用，这让我对人工智能有了初步的认识。今年暑假期间，我阅读了尼克的《人工智能简史》，这本书讲述了人工智能学科各领域的发展历史和轶事，进一步增加了我对人工智能的兴趣。于是，这学期我选择了《人工智能导论》这门课，希望能加深对人工智能的认识和理解。一学期的时间过得很快，转眼课程已进入尾声。下面总结一下我在这一学期的课程中的收获。

**一、课堂学习方面**

在前3周课程中，我了解了人工智能发展的历史和现状，学习了传统AI方法中的推理技术和搜索技术。之后的课堂学习主要依托于老师推荐我们使用的参考书《Hands-on Machine Learning with Scikit-Learn & TensorFlow》，这本书主要分为两部分：第一部分为机器学习基础，主要基于Scikit-Learn；第二部分为神经网络与深度学习，主要基于TensorFlow。

根据课程所学和课外阅读，我认为人工智能与机器学习是一种包含关系（如下图1），机器学习所做的事情其实是一种模式识别，它属于人工智能的第二个发展阶段。在人工智能的第一个发展阶段，人们试图使用逻辑和规则直接来演绎人类的智能，这个时期出现了很多基于规则的专家系统，很快这种方式被证明是走不通的，而出现了让机器真正学会人类学习方式的范式——机器学习。机器学习返回了智慧产生的本源（学习），通过大量的数据归纳出统计的规律。

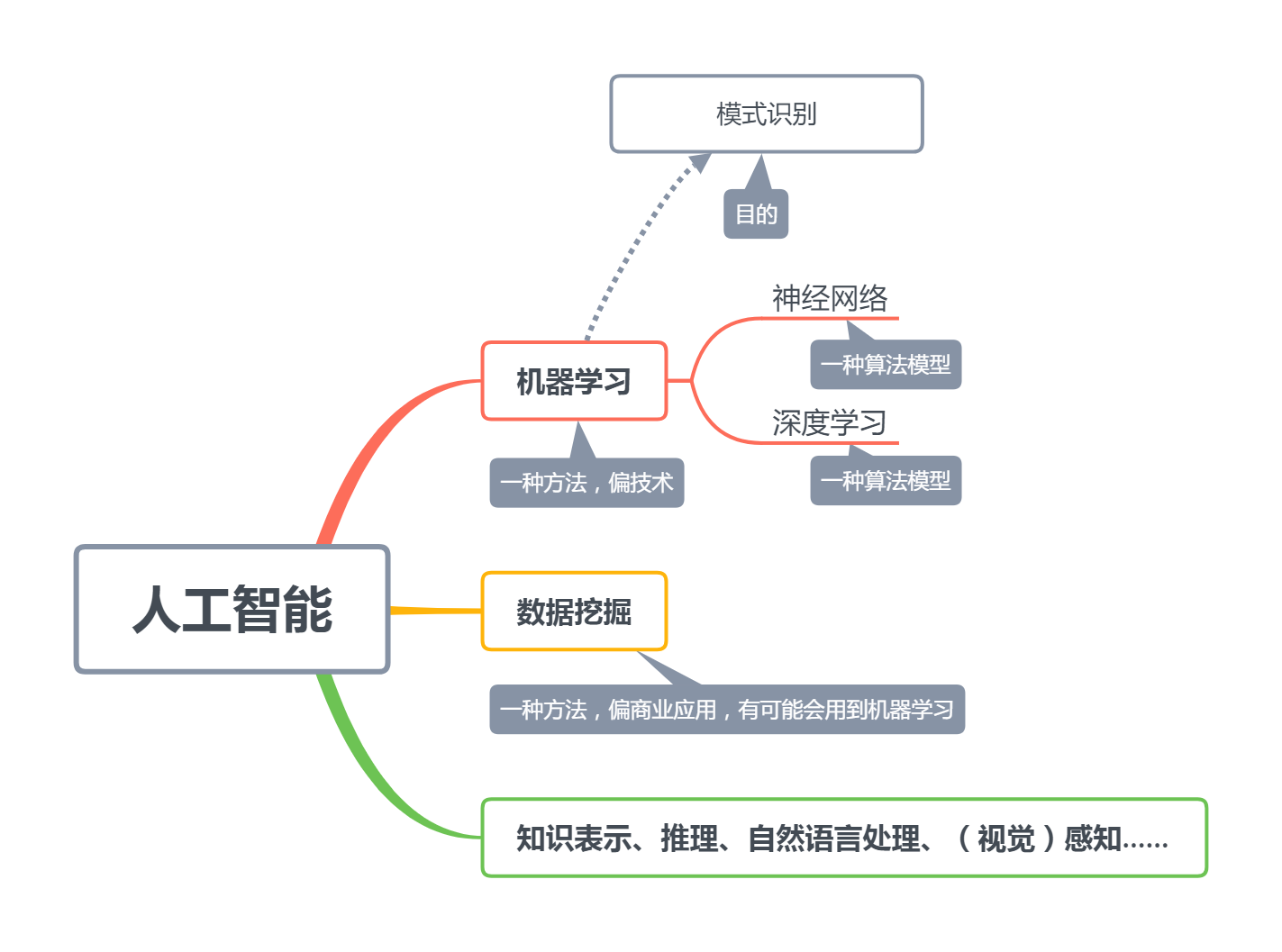


图 1（人工智能与机器学习的关系）

根据训练时监督的量和类型进行分类，机器学习主要可分为四类：监督学习、非监督学习、半监督学习和强化学习。如下图所示：

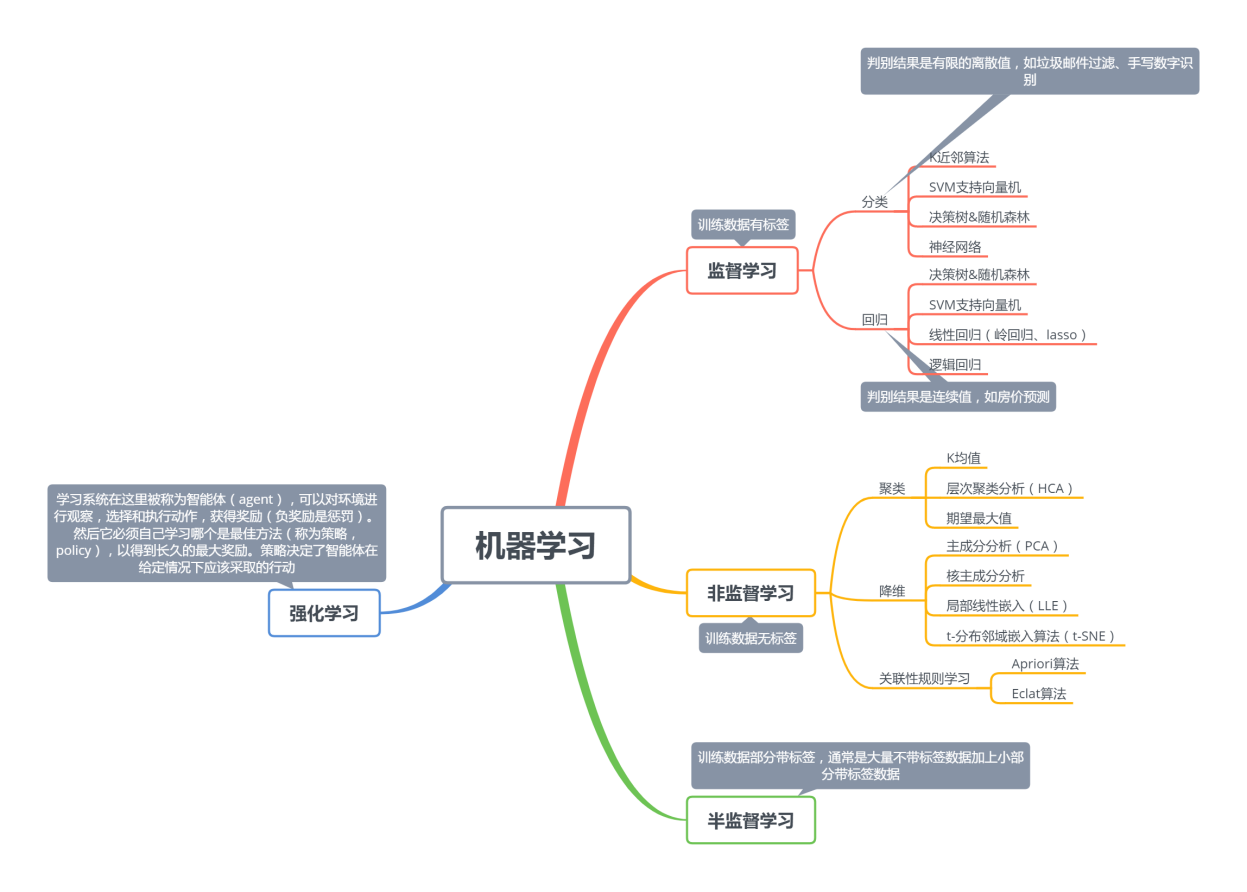


图 2（机器学习的第一种分类）

根据能否从导入的数据流进行持续学习进行分类，机器学习可分为：批量学习和在线学习。在批量学习中，系统不能进行持续学习：必须用所有可用数据进行训练，这通常会占用大量时间和计算资源，所以一般是线下做的。首先是进行训练，然后部署在生产环境且停止学习，它只是使用已经学到的策略，这称为离线学习；在在线学习中，系统用数据实例持续地进行训练，可以一次一个或一次几个实例，系统可以动态地学习到达的数据。

根据泛化方式进行分类，机器学习可分为：基于实例学习和基于模型学习。基于实例学习是：先用记忆学习案例，然后用相似度测量推广到新的例子；基于模型学习是：建立这些样本的模型，然后使用模型进行预测。参考书中第3章及之后的内容主要按照第一种分类方式展开。

通过阅读学习参考书，我熟悉了一个完整的机器学习项目的主要步骤：

1. 总览问题和目标（Look at the big picture）
2. 获取数据（Get the data）
3. 探索数据（Discover and visualize the data to gain insights）
4. 为模型训练预处理数据（Prepare the data for Machine Learning algorithms）
5. 选择不同的模型加以训练，得到最好的一个或几个（Select a model and train it）
6. 调节模型(超参数), 有必要时进行模型组合（Fine-tune your model）
7. 演示模型（Present your solution）
8. 启动，监视和维护模型（Launch, monitor, and maintain your system）。

我觉得在调节模型方面，自己还应该加强学习。

**二、小组作业方面**

本学期的小组作业分为三个阶段，目前我们已经完成了两个阶段。

在第一阶段，我们小组的主要成果是：

1. 完成了Arduino小车硬件的组装；
2. 实现了小车的基本动作：前进、停止、左转、右转和后退；
3. 实现了三大基础功能：循迹（红外循迹）、避障（红外避障与超声波避障）、遥控（蓝牙遥控）；
4. 追踪特定物体：通过“迹”app识别屏幕中特定颜色的物体，并通过蓝牙串口返回物体坐标值，程序根据坐标判断前进/后退方向。

在第二阶段，我们小组的主要成果是：

1. 图像识别方面：基于OpenCV、Ulib和Face\_Recognition分别对白敬亭、魏大勋的静态照片和实时视频进行人脸识别；
2. 语音控制方面：基于DTW和GMM-HMM识别“前”、“后”、“左”、“右”、“停”五个指令，通过电脑蓝牙发送给小车蓝牙，从而控制小车的运动。

在第一阶段，我主要参与了小车基本动作及三大基础功能的实现，制作了与这部分相关的PPT内容，并在展示中报告了这部分的成果；在第二阶段，我主要参与了语音控制部分的实现，制作了与这部分相关的PPT内容，并在展示中报告了这部分成果。

在完成小组作业期间，我们遇到了不少问题：比如刚开始电脑识别不出Arduino开发板的端口，无法将程序成功上传，后来在实践中发现原因是Arduino板上连线过多，导致电流不足。于是我们在上传程序时，将电源打开为Arduino供电，成功解决该问题；还有在蓝牙通信时，曾遇到电脑搜索不到Arduino的蓝牙的情况，后来查阅资料得知：一对设备进行蓝牙通信时，其中一个为主角色，一个为从角色，在蓝牙配对后，只有主端删除设备，从端才能被其他主端搜索到。

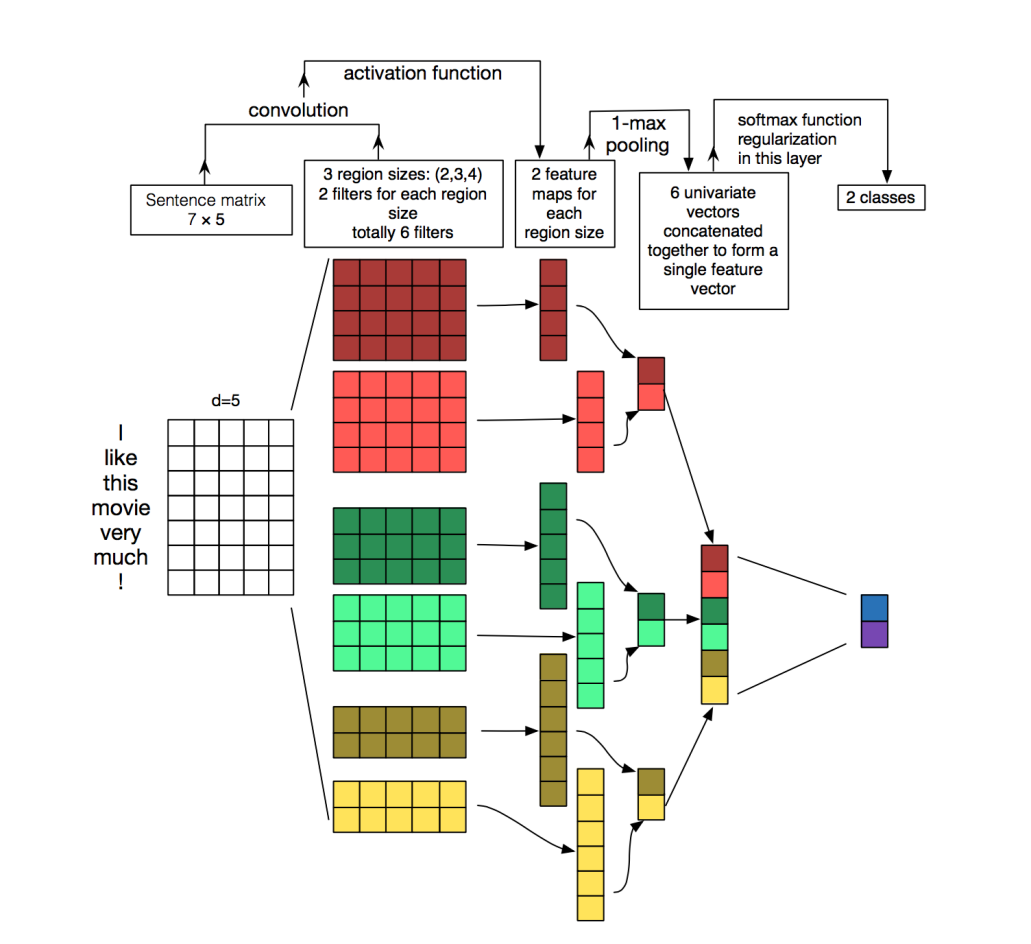
在发现问题与解决问题的过程中，我掌握了Arduino小车的硬件搭建方法，熟悉了手机蓝牙、电脑蓝牙与Arduino蓝牙串口的通信方式，能够根据教程编写基本的ino代码。同时，我将课堂上学到的知识学以致用，如在小车的语音控制上应用了言语信息处理课上学过的DTW算法和GMM-HMM模型，在小车的图像识别方面应用了tensorFlow训练的神经网络。

**三、课下实践方面**

课下我在万门大学上面报名了《机器学习原理与应用入门》和《深度学习与神经网络》两门课，同时参考阅读了邱锡鹏的《神经网络与深度学习》。这些课程和书籍使我进一步加深了对课堂所学知识的理解。

另外，为了练习TensorFlow的使用，我参考Kim Yoon的论文、Denny Britz的教程Implementing a CNN for Text Classification in TensorFlow，尝试使用CNN对TREC06C中文邮件数据集进行中文文本分类。

CNN通常应用于图像处理，图像输入相当于一个二维数组，使用的卷积函数为二维卷积函数。类似地，对于文本数据的处理，文本输入相当于一个一维数组，使用的卷积函数应为一维卷积函数。教程中给出了使用CNN处理文本数据的一般过程，如下所示：



图片来源：Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification.

① 输入为7×5大小的文本。

② 滤波器的大小有2，3，4三种，每种大小的滤波器有2个，一共6个滤波器，每个滤波器对文本矩阵进行卷积并生成特征图。大小为2的滤波器生成的特征图大小为6，大小为3的滤波器生成的特征图大小为5，大小为4的滤波器生成的特征图大小为4。

③ 对每个特征图进行1-max池化，即记录每个特征图的最大值，因此6个特征图生成了6个一元向量，这6个一元向量连接起来组成了倒数第二层的特征向量。

④ 最后的softmax层接收倒数第二层的特征向量作为输入，用它来给文本分类，最终得到两种可能的输出状态（我们的文本分类为二分类）。

与上述过程相比，我增加了对中文数据集的预处理过程，包括过滤非中文字符、分词填充、训练word2vec模型。我用word2vec训练出的词向量代替了上图中的文本矩阵作为输入。

以下是使用的代码及分析：

**（一）中文邮件数据集的预处理**

**1. 数据过滤**

def read\_and\_clean\_zh\_file(input\_file, output\_cleaned\_file = None):

lines = list(open(input\_file, "rb").readlines())

lines = [clean\_str(seperate\_line(line.decode('utf-8'))) for line in lines]

if output\_cleaned\_file is not None:

with open(output\_cleaned\_file, 'w') as f:

for line in lines:

f.write((line + '\n').encode('utf-8'))

return lines

def clean\_str(string):

string = re.sub(r"[^\u4e00-\u9fff]", " ", string)

string = re.sub(r"\s{2,}", " ", string)

return string.strip()

读入原数据，过滤掉数据中的所有非中文字符。

**2. 分词及填充**

def padding\_sentences(input\_sentences, padding\_token, padding\_sentence\_length = None):

sentences = [sentence.split(' ') for sentence in input\_sentences]

max\_sentence\_length = padding\_sentence\_length if padding\_sentence\_length is not None else max([len(sentence) for sentence in sentences])

for sentence in sentences:

if len(sentence) > max\_sentence\_length:

sentence = sentence[:max\_sentence\_length]

else:

sentence.extend([padding\_token] \* (max\_sentence\_length - len(sentence)))

return (sentences, max\_sentence\_length)

简便起见，对每个中文字符进行分隔，将所有邮件填充至长度为max\_sentence\_length（最长邮件的长度）。

**3. 训练word2vec模型**

def embedding\_sentences(sentences, embedding\_size = 128, window = 5, min\_count = 5, file\_to\_load = None, file\_to\_save = None):

if file\_to\_load is not None:

w2vModel = Word2Vec.load(file\_to\_load)

else:

w2vModel = Word2Vec(sentences, size = embedding\_size, window = window, min\_count = min\_count, workers = multiprocessing.cpu\_count())

if file\_to\_save is not None:

w2vModel.save(file\_to\_save)

all\_vectors = []

embeddingDim = w2vModel.vector\_size

embeddingUnknown = [0 for i in range(embeddingDim)]

for sentence in sentences:

this\_vector = []

for word in sentence:

if word in w2vModel.wv.vocab:

this\_vector.append(w2vModel[word])

else:

this\_vector.append(embeddingUnknown)

all\_vectors.append(this\_vector)

return all\_vectors

**4. 得到x，y**

x是所有邮件的向量表示，[所有邮件个数，max\_doument\_length，embedding\_dim]。

y是所有邮件对应的标签，[所有邮件个数，2]，对第二个维度，[0，1]表示正常邮件，[1，0]表示垃圾邮件。

def load\_data\_and\_labels(input\_text\_file, input\_label\_file, num\_labels):

x\_text = read\_and\_clean\_zh\_file(input\_text\_file)

y = None if not os.path.exists(input\_label\_file) else map(int, list(open(input\_label\_file, "r").readlines()))

return (x\_text, y)

def load\_positive\_negative\_data\_files(positive\_data\_file, negative\_data\_file):

"""

Loads MR polarity data from files, splits the data into words and generates labels.

Returns split sentences and labels.

"""

# Load data from files

positive\_examples = read\_and\_clean\_zh\_file(positive\_data\_file)

negative\_examples = read\_and\_clean\_zh\_file(negative\_data\_file)

# Combine data

x\_text = positive\_examples + negative\_examples

# Generate labels

positive\_labels = [[0, 1] for \_ in positive\_examples]

negative\_labels = [[1, 0] for \_ in negative\_examples]

y = np.concatenate([positive\_labels, negative\_labels], 0)

return [x\_text, y]

**（二）定义CNN网络**

**1. 在TextCNN类的init函数中配置各种超参数接口**

import tensorflow as tf

import numpy as np

class TextCNN(object):

'''

A CNN for text classification

Uses and embedding layer, followed by a convolutional, max-pooling and softmax layer.

'''

def \_\_init\_\_(

self, sequence\_length, num\_classes,

embedding\_size, filter\_sizes, num\_filters, l2\_reg\_lambda=0.0):

之后实例化这个类时，我传递了如下的参数：

sequence\_length - 填充后的邮件长度（实质是最长邮件的长度），190。

num\_classes - 输出层的类数，2（正常邮件ham和垃圾邮件spam）。

embedding\_size - word2vec模型训练出的embedding向量维数，128。

filter\_sizes - 滤波器的大小，即滤波器覆盖的字数，[3, 4, 5]。

num\_filters - 每种大小的滤波器的数量，128。

**2. 定义传递给网络的输入数据**

# Placeholders for input, output, dropout

self.input\_x = tf.placeholder(tf.float32, [None, sequence\_length, embedding\_size], name = "input\_x")

self.input\_y = tf.placeholder(tf.float32, [None, num\_classes], name = "input\_y")

self.dropout\_keep\_prob = tf.placeholder(tf.float32, name = "dropout\_keep\_prob")

# Keeping track of l2 regularization loss (optional)

l2\_loss = tf.constant(0.0)

tf.placeholder创建一个占位符变量，在训练或测试中执行该变量时，向网络提供该变量。第二个参数是输入的tensor，其第一个维度为None，表示允许网络处理任意大小的batch。

**3. 将word2vec预训练出的embedding向量输入到网络中**

# Embedding layer

self.embedded\_chars = self.input\_x

self.embedded\_chars\_expended = tf.expand\_dims(self.embedded\_chars, -1)

**4. 卷积层和最大值池化层**

# Create a convolution + maxpool layer for each filter size

pooled\_outputs = []

for i, filter\_size in enumerate(filter\_sizes):

with tf.name\_scope("conv-maxpool-%s" % filter\_size):

# Convolution layer

filter\_shape = [filter\_size, embedding\_size, 1, num\_filters]

W = tf.Variable(tf.truncated\_normal(filter\_shape, stddev=0.1), name="W")

b = tf.Variable(tf.constant(0.1, shape=[num\_filters]), name="b")

conv = tf.nn.conv2d(self.embedded\_chars\_expended,

W,

strides=[1,1,1,1],

padding="VALID",

name="conv")

# Apply nonlinearity

h = tf.nn.relu(tf.nn.bias\_add(conv, b), name = "relu")

# Maxpooling over the outputs

pooled = tf.nn.max\_pool(

h,

ksize=[1, sequence\_length - filter\_size + 1, 1, 1],

strides=[1,1,1,1],

padding="VALID",

name="pool")

pooled\_outputs.append(pooled)

# Combine all the pooled features

num\_filters\_total = num\_filters \* len(filter\_sizes)

self.h\_pool = tf.concat(pooled\_outputs, 3)

self.h\_pool\_flat = tf.reshape(self.h\_pool, [-1, num\_filters\_total])

此处W是滤波器矩阵，h是在卷积后进行非线性操作的输出结果。

padding = "VALID"表示滤波器滑动时对边缘无填充，执行一个窄卷积，从而产生形如[1, sequence\_length - filter\_size + 1, 1, 1]的输出。

然后对每种尺寸的滤波器的输出进行max-pooling，产生形如[batch\_size, 1, 1, num\_filters]的特征向量，其最后一个维度对应于学到的数据特征。

最后将池化后产生的所有特征向量合并为一个长特征向量[batch\_size, num\_filters\_total]。

**5. Dropout层**

# Add dropout

with tf.name\_scope("dropout"):

self.h\_drop = tf.nn.dropout(self.h\_pool\_flat, self.dropout\_keep\_prob)

定义Dropout层防止CNN过拟合，训练过程中将dropout\_keep\_prob参数的值设置为0.5。

**6. 分数及预测**

# Final (unnomalized) scores and predictions

with tf.name\_scope("output"):

W = tf.get\_variable("W",

shape = [num\_filters\_total, num\_classes],

initializer = tf.contrib.layers.xavier\_initializer())

b = tf.Variable(tf.constant(0.1, shape=[num\_classes], name = "b"))

l2\_loss += tf.nn.l2\_loss(W)

l2\_loss += tf.nn.l2\_loss(b)

self.scores = tf.nn.xw\_plus\_b(self.h\_drop, W, b, name = "scores")

self.predictions = tf.argmax(self.scores, 1, name = "predictions")

使用max-pooling中的特征向量，通过矩阵乘法生成预测，并选择得分最高的类。

**7. 损失和精度**

# Calculate Mean cross-entropy loss

with tf.name\_scope("loss"):

losses = tf.nn.softmax\_cross\_entropy\_with\_logits(logits = self.scores, labels = self.input\_y)

self.loss = tf.reduce\_mean(losses) + l2\_reg\_lambda \* l2\_loss

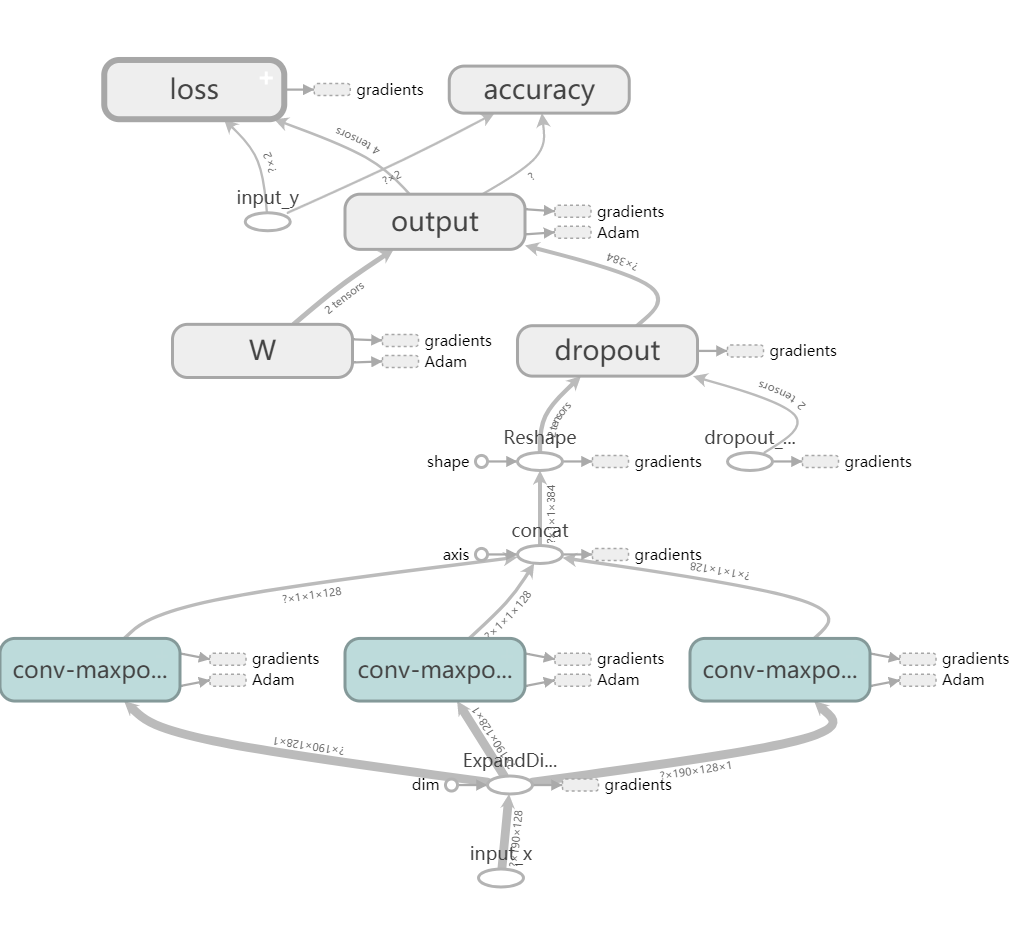
# Accuracy

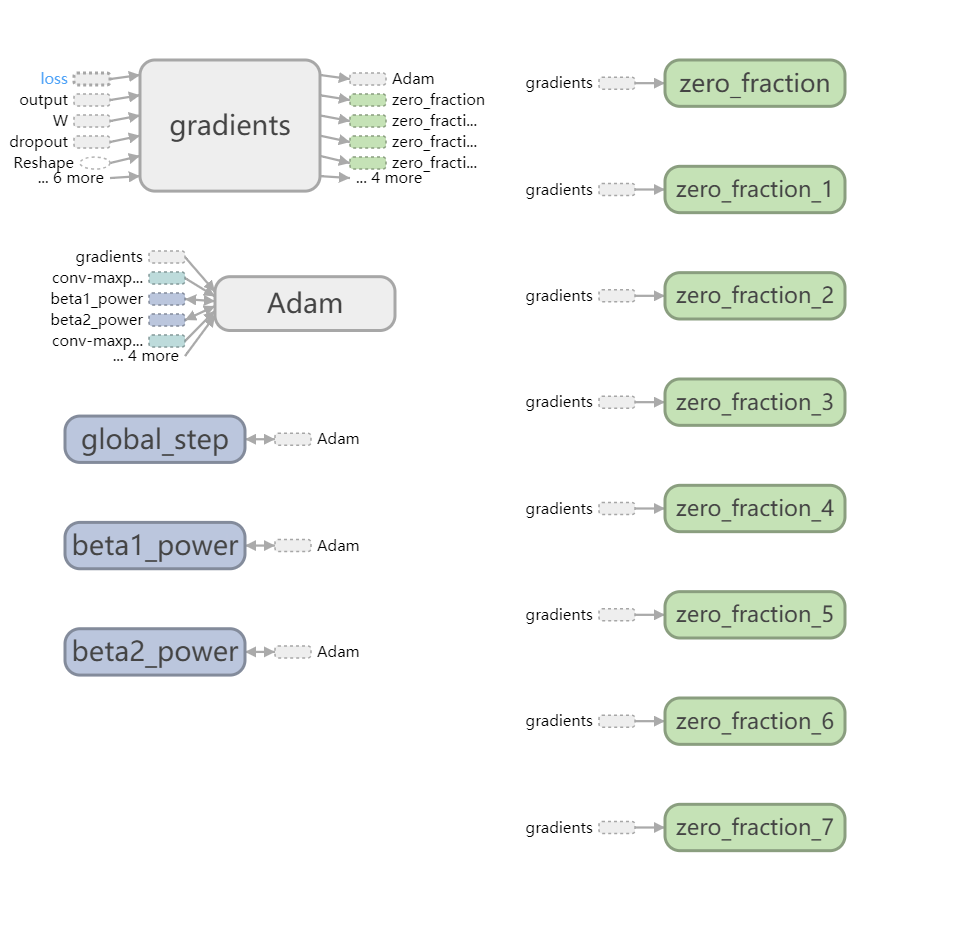
with tf.name\_scope("accuracy"):

correct\_predictions = tf.equal(self.predictions, tf.argmax(self.input\_y, 1))

self.accuracy = tf.reduce\_mean(tf.cast(correct\_predictions, "float"), name = "accuracy")

最终定义完的CNN结构如下图所示：





1. **训练过程**

**1. CNN实例化**

cnn = TextCNN(

sequence\_length = x\_train.shape[1],

num\_classes = y\_train.shape[1],

embedding\_size = FLAGS.embedding\_dim,

filter\_sizes = list(map(int, FLAGS.filter\_sizes.split(","))),

num\_filters = FLAGS.num\_filters,

l2\_reg\_lambda = FLAGS.l2\_reg\_lambda)

**2. 初始化变量**

# Initialize all variables

sess.run(tf.global\_variables\_initializer())

**3. 定义train step**

def train\_step(x\_batch, y\_batch):

"""

A single training step

"""

feed\_dict = {

cnn.input\_x: x\_batch,

cnn.input\_y: y\_batch,

cnn.dropout\_keep\_prob: FLAGS.dropout\_keep\_prob

}

\_, step, summaries, loss, accuracy = sess.run(

[train\_op, global\_step, train\_summary\_op, cnn.loss, cnn.accuracy],

feed\_dict)

time\_str = datetime.datetime.now().isoformat()

print("{}: step {}, loss {:g}, acc {:g}".format(time\_str, step, loss, accuracy))

train\_summary\_writer.add\_summary(summaries, step)

**4. 定义dev step**

def dev\_step(x\_batch, y\_batch, writer=None):

"""

Evaluates model on a dev set

"""

feed\_dict = {

cnn.input\_x: x\_batch,

cnn.input\_y: y\_batch,

cnn.dropout\_keep\_prob: 1.0

}

step, summaries, loss, accuracy = sess.run(

[global\_step, dev\_summary\_op, cnn.loss, cnn.accuracy],

feed\_dict)

time\_str = datetime.datetime.now().isoformat()

print("{}: step {}, loss {:g}, acc {:g}".format(time\_str, step, loss, accuracy))

if writer:

writer.add\_summary(summaries, step)

**5. 训练循环**

# Generate batches

batches = data\_helpers.batch\_iter(

list(zip(x\_train, y\_train)), FLAGS.batch\_size, FLAGS.num\_epochs)

# Training loop. For each batch...

for batch in batches:

x\_batch, y\_batch = zip(\*batch)

train\_step(x\_batch, y\_batch)

current\_step = tf.train.global\_step(sess, global\_step)

if current\_step % FLAGS.evaluate\_every == 0:

print("\nEvaluation:")

dev\_step(x\_dev, y\_dev, writer=dev\_summary\_writer)

print("")

if current\_step % FLAGS.checkpoint\_every == 0:

path = saver.save(sess, checkpoint\_prefix, global\_step=current\_step)

print("Saved model checkpoint to {}\n".format(path))

**（四）评估过程**

**1. 对得到的checkpoint进行评估**

print("\nEvaluating...\n")

checkpoint\_file = tf.train.latest\_checkpoint(FLAGS.checkpoint\_dir)

graph = tf.Graph()

with graph.as\_default():

session\_conf = tf.ConfigProto(

allow\_soft\_placement=FLAGS.allow\_soft\_placement,

log\_device\_placement=FLAGS.log\_device\_placement)

sess = tf.Session(config=session\_conf)

with sess.as\_default():

# Load the saved meta graph and restore variables

saver = tf.train.import\_meta\_graph("{}.meta".format(checkpoint\_file))

saver.restore(sess, checkpoint\_file)

# Get the placeholders from the graph by name

input\_x = graph.get\_operation\_by\_name("input\_x").outputs[0]

# input\_y = graph.get\_operation\_by\_name("input\_y").outputs[0]

dropout\_keep\_prob = graph.get\_operation\_by\_name("dropout\_keep\_prob").outputs[0]

# Tensors we want to evaluate

predictions = graph.get\_operation\_by\_name("output/predictions").outputs[0]

# Generate batches for one epoch

batches = data\_helpers.batch\_iter(list(x\_test), FLAGS.batch\_size, 1, shuffle=False)

# Collect the predictions here

all\_predictions = []

for x\_test\_batch in batches:

batch\_predictions = sess.run(predictions, {input\_x: x\_test\_batch, dropout\_keep\_prob: 1.0})

all\_predictions = np.concatenate([all\_predictions, batch\_predictions])

**2. 将评估结果保存到prediction.csv**

# Save the evaluation to a csv

predictions\_human\_readable = np.column\_stack((np.array([text.encode('utf-8').decode() for text in x\_raw]), all\_predictions))

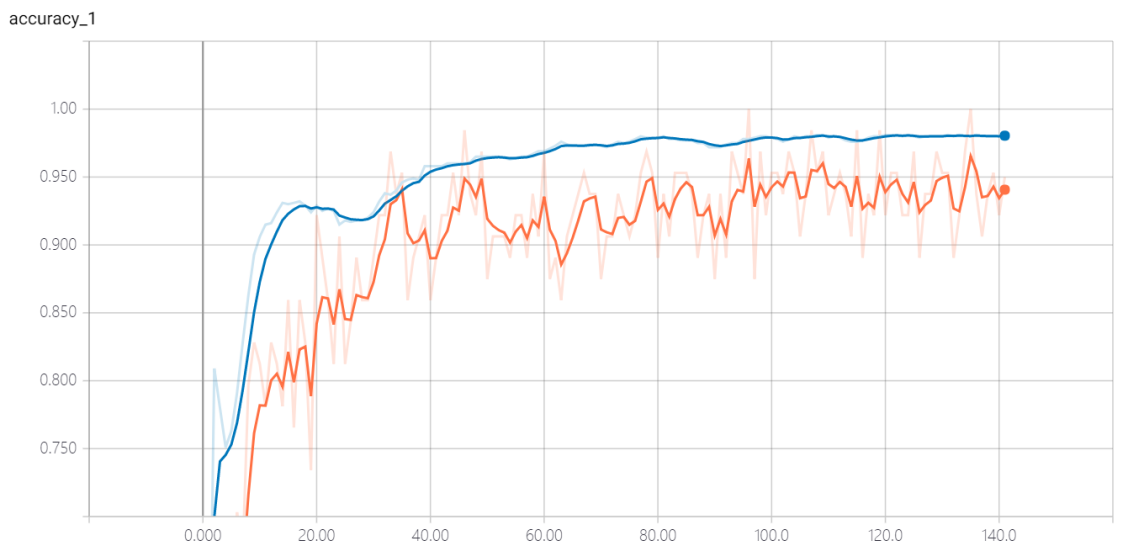
out\_path = os.path.join(FLAGS.checkpoint\_dir, "..", "prediction.csv")

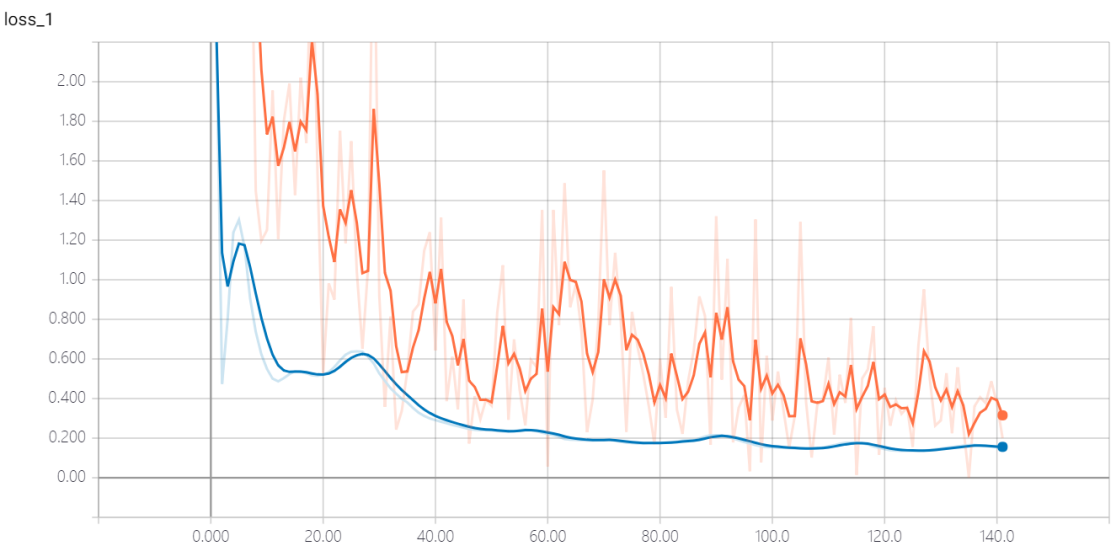
print("Saving evaluation to {0}".format(out\_path))

with open(out\_path, 'w') as f:

csv.writer(f).writerows(predictions\_human\_readable)

实验中以10%的数据作为测试集，在tensorboard中的准确度与损失可视化展示如下图，其中橙色代表训练集train，蓝色代表测试集dev：





由上图可看出，在100 step之后准确度基本在95%左右。