一文详解 Word2vec 之 Skip-Gram 模型（实现篇）

本文作者：AI研习社 2017-06-24 10:15

导语：这可能是关于 Skip-Gram 模型最详细的讲解。

雷锋网按：这是一个关于 Skip-Gram 模型的系列教程，共分为结构、训练和实现三个部分，本文为最后一部分：实现篇。原文作者天雨粟，原载于作者知乎专栏，雷锋网已获授权。

前言

上一篇的专栏介绍了Word2Vec中的Skip-Gram模型的结构和训练，如果看过的小伙伴可以直接开始动手用TensorFlow实现自己的Word2Vec模型，本篇文章将利用TensorFlow来完成Skip-Gram模型。还不是很了解Skip-Gram思想的小伙伴可以先看一下上一篇的专栏内容。

本篇实战代码的目的主要是加深对Skip-Gram模型中一些思想和trick的理解。由于受限于语料规模、语料质量、算法细节以及训练成本的原因，训练出的结果显然是无法跟gensim封装的Word2Vec相比的，本代码适合新手去理解与练习Skip-Gram模型的思想。

工具介绍

- 语言：Python 3
- 包：TensorFlow（1.0版本）及其它数据处理包（见代码中）
- 编辑器：jupyter notebook
- 线上GPU：floyd (https://www.floydhub.com/)
- 数据集：经过预处理后的维基百科文章（英文）

正文部分

文章主要包括以下四个部分进行代码构造：

> - 数据预处理
> - 训练样本构建

- 模型构建
- 模型验证

1 数据预处理

关于导入包和加载数据在这里就不写了，比较简单，请参考git上的代码。

数据预处理部分主要包括：

- 替换文本中特殊符号并去除低频词
- 对文本分词
- 构建语料
- 单词映射表

首先我们定义一个函数来完成前两步，即对文本的清洗和分词操作。

```
# 定义函数来完成数据的预处理
def preprocess(text, freq=5):
    '''
    对文本进行预处理

    参数
    ---
    text: 文本数据
    freq: 词频阈值
    '''
    # 对文本中的符号进行替换
    text = text.lower()
    text = text.replace('.', ' <PERIOD> ')
    text = text.replace(',', ' <COMMA> ')
    text = text.replace('"', ' <QUOTATION_MARK> ')
    text = text.replace(';', ' <SEMICOLON> ')
    text = text.replace('!', ' <EXCLAMATION_MARK> ')
    text = text.replace('?', ' <QUESTION_MARK> ')
    text = text.replace('(', ' <LEFT_PAREN> ')
    text = text.replace(')', ' <RIGHT_PAREN> ')
    text = text.replace('--', ' <HYPHENS> ')
    text = text.replace('?', ' <QUESTION_MARK> ')
    # text = text.replace('\n', ' <NEW_LINE> ')
    text = text.replace(':', ' <COLON> ')
    words = text.split()

    # 删除低频词，减少噪音影响
    word_counts = Counter(words)
    trimmed_words = [word for word in words if word_counts[word] > freq]

    return trimmed_words
```

上面的函数实现了替换标点及删除低频词操作，返回分词后的文本。

下面让我们来看看经过清洗后的数据：

```
# 清洗文本并分词
words = preprocess(text)
print(words[:20])
```

```
['anarchism', 'originated', 'as', 'a', 'term', 'of', 'abuse', 'first', 'u
sed', 'against', 'early', 'working', 'class', 'radicals', 'including', 't
he', 'diggers', 'of', 'the', 'english']
```

有了分词后的文本，就可以构建我们的映射表，代码就不再赘述，大家应该都比较熟悉。

```
# 构建映射表
vocab = set(words)
vocab_to_int = {w: c for c, w in enumerate(vocab)}
int_to_vocab = {c: w for c, w in enumerate(vocab)}
```

我们还可以看一下文本和词典的规模大小：

```
print("total words: {}".format(len(words)))
print("unique words: {}".format(len(set(words))))
```

```
total words: 16680599
unique words: 63641
```

百度视觉团队斩获 ECCV Go
AI 目标检测竞赛冠军，获奖
全解读 | ECCV 2018

清华大学王延森：如何利用
增强开放领域对话系统互动
AI研习社66期大讲堂

如何从静态图像中识别“比
心”动作

ICPR 图像识别与检测挑战赛
方案出炉，基于偏旁部首来
Duang 字

热门搜索

Uber 神经网络
印度 Android游戏
Yann LeCun Kindle Fi
酷派 CPU 智能
硬件创业 国产手机

整个文本中单词大约为1660万的规模，词典大小为6万左右，这个规模对于训练好的词向量其实是不够的，但可以训练出一个稍微还可以的模型。

2 训练样本构建

我们知道skip-gram中，训练样本的形式是(input word, output word)，其中output word是input word的上下文。为了减少模型噪音并加速训练速度，我们在构造batch之前要对样本进行采样，剔除停用词等噪音因素。

采样

在建模过程中，训练文本中会出现很多“the”、“a”之类的常用词（也叫停用词），这些词对于我们的训练会带来很多噪音。在上一篇Word2Vec中提过对样本进行抽样，剔除高频的停用词来减少模型的噪音，并加速训练。

我们采用以下公式来计算每个单词被删除的概率大小：

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

其中 $f(w_i)$ 代表单词 w_i 的出现频次。t为一个阈值，一般介于1e-3到1e-5之间。

```
t = 1e-5 # t值
threshold = 0.8 # 剔除概率阈值

# 统计单词出现频次
int_word_counts = Counter(int_words)
total_count = len(int_words)
# 计算单词频率
word_freqs = {w: c/total_count for w, c in int_word_counts.items()}
# 计算被删除的概率
prob_drop = {w: 1 - np.sqrt(t / f) for w, f in word_freqs.items()}
# 对单词进行采样
train_words = [w for w in int_words if prob_drop[w] < threshold]
```

上面的代码计算了样本中每个单词被删除的概率，并基于概率进行了采样，现在我们手里就拿到了采样过的单词列表。

构造batch

我们先来分析一下skip-gram的样本格式。skip-gram不同于CBOW，CBOW是基于上下文预测当前input word。而skip-gram则是基于一个input word来预测上下文，因此一个input word会对应多个上下文。我们来举个栗子“The quick brown fox jumps over lazy dog”，如果我们固定skip_window=2的话，那么fox的上下文就是[quick, brown, jumps, over]，如果我们的batch_size=1的话，那么实际上一个batch中有四个训练样本。

上面的分析转换为代码就是两个步骤，第一个是找到每个input word的上下文，第二个就是基于上下文构建batch。

首先是找到input word的上下文单词列表：

```
def get_targets(words, idx, window_size=5):
    '''
    获得input word的上下文单词列表

    参数
    ---
    words: 单词列表
    idx: input word的索引号
    window_size: 窗口大小
    '''
    target_window = np.random.randint(1, window_size+1)
    # 这里要考虑input word前面单词不够的情况
    start_point = idx - target_window if (idx - target_window) > 0 else 0
    end_point = idx + target_window
    # output words(即窗口中的上下文单词)
    targets = set(words[start_point: idx] + words[idx+1: end_point+1])
    return list(targets)
```

我们定义了一个get_targets函数，接收一个单词索引号，基于这个索引号去查找单词表中对应的上下文（默认window_size=5）。请注意这里有一个小trick，我在实际选择input word上下文时，使用的窗口

大小是一个介于[1, window_size]区间的随机数。这里的目的是让模型更多地去关注离input word更近词。

我们有了上面的函数后，就能够轻松地通过input word找到它的上下文单词。有了这些单词我们就可以构建我们的batch来进行训练：

```
def get_batches(words, batch_size, window_size=5):
    '''
    构造一个获取batch的生成器
    '''
    n_batches = len(words) // batch_size

    # 仅取full batches
    words = words[:n_batches*batch_size]

    for idx in range(0, len(words), batch_size):
        x, y = [], []
        batch = words[idx: idx+batch_size]
        for i in range(len(batch)):
            batch_x = batch[i]
            batch_y = get_targets(batch, i, window_size)
            # 由于一个input word会对应多个output word, 因此需要长度统一
            x.extend([batch_x]*len(batch_y))
            y.extend(batch_y)
        yield x, y
```

注意上面的代码对batch的处理。我们知道对于每个input word来说，有多个output word（上下文）。例如我们的输入是“fox”，上下文是[quick, brown, jumps, over]，那么fox这一个batch中就有四个训练样本[fox, quick], [fox, brown], [fox, jumps], [fox, over]。

3 模型构建

数据预处理结束后，就需要来构建我们的模型。在模型中为了加速训练并提高词向量的质量，我们采用负采样方式进行权重更新。

输入层到嵌入层

输入层到隐层的权重矩阵作为嵌入层要给定其维度，一般embeding_size设置为50-300之间。

```
vocab_size = len(int_to_vocab)
embedding_size = 200 # 嵌入维度
```

```
with train_graph.as_default():
    # 嵌入层权重矩阵
    embedding = tf.Variable(tf.random_uniform([vocab_size, embedding_size], -1, 1))
    # 实现lookup
    embed = tf.nn.embedding_lookup(embedding, inputs)
```

嵌入层的 lookup 通过 TensorFlow 中的 embedding_lookup 实现，详见：

http://t.cn/RofvbgF

嵌入层到输出层

在skip-gram中，每个input word的多个上下文单词实际上是共享一个权重矩阵，我们将每个（input word, output word）训练样本来作为我们的输入。为了加速训练并且提高词向量的质量，我们采用negative sampling的方法来进行权重更新。

TensorFlow中的sampled_softmax_loss，由于进行了negative sampling，所以实际上我们会低估模型的训练loss。详见：http://t.cn/RofvS4t

```
n_sampled = 100

with train_graph.as_default():
    softmax_w = tf.Variable(tf.truncated_normal([vocab_size, embedding_size], stddev=0.1))
    softmax_b = tf.Variable(tf.zeros(vocab_size))

    # 计算negative sampling下的损失
    loss = tf.nn.sampled_softmax_loss(softmax_w, softmax_b, labels, embed, n_sampled, vocab_size)

    cost = tf.reduce_mean(loss)
    optimizer = tf.train.AdamOptimizer().minimize(cost)
```

> 请注意代码中的softmax_w的维度是vocab_size x embedding_size，这是因为TensorFlow中的sampled_softmax_loss中参数weights的size是[num_classes, dim]。

4 模型验证

在上面的步骤中，我们已经将模型的框架搭建出来，下面就让我们来训练训练一下模型。为了能够更加直观地观察训练每个阶段的情况。我们来挑选几个词，看看在训练过程中它们的相似词是怎么变化的。

```
with train_graph.as_default():
    # 随机挑选一些单词
    valid_size = 16
    valid_window = 100
    # 从不同位置各选8个单词
    valid_examples = np.array(random.sample(range(valid_window), valid_size//2))
    valid_examples = np.append(valid_examples,
                               random.sample(range(1000,1000+valid_window), valid_size//2))

    valid_size = len(valid_examples)
    # 验证单词集
    valid_dataset = tf.constant(valid_examples, dtype=tf.int32)

    # 计算每个词向量的模并进行单位化
    norm = tf.sqrt(tf.reduce_sum(tf.square(embedding), 1, keep_dims=True))
    normalized_embedding = embedding / norm
    # 查找验证单词的词向量
    valid_embedding = tf.nn.embedding_lookup(normalized_embedding, valid_dataset)
    # 计算余弦相似度
    similarity = tf.matmul(valid_embedding, tf.transpose(normalized_embedding))
```

训练模型：

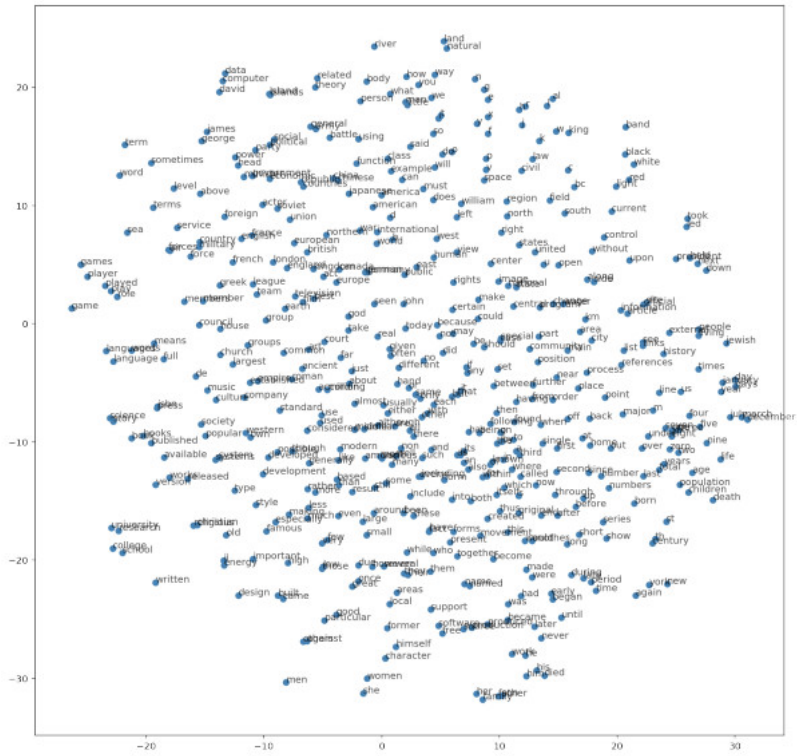
```
epochs = 10 # 迭代轮数
batch_size = 1000 # batch大小
window_size = 10 # 窗口大小

with train_graph.as_default():
    saver = tf.train.Saver() # 文件存储

with tf.Session(graph=train_graph) as sess:
    iteration = 1
    loss = 0
    sess.run(tf.global_variables_initializer())

    for e in range(1, epochs+1):
        batches = get_batches(train_words, batch_size, window_size)
        start = time.time()
        #
        for x, y in batches:

            feed = {inputs: x,
                    labels: np.array(y)[:, None]}
            train_loss, _ = sess.run([cost, optimizer], feed_dict=feed)

            loss += train_loss

            if iteration % 100 == 0:
                end = time.time()
                print("Epoch {}/{}".format(e, epochs),
                      "Iteration: {}".format(iteration),
                      "Avg. Training loss: {:.4f}".format(loss/100),
                      "{:.4f} sec/batch".format((end-start)/100))
                loss = 0
                start = time.time()

            # 计算相似的词
            if iteration % 1000 == 0:
                # 计算similarity
                sim = similarity.eval()
                for i in range(valid_size):
                    valid_word = int_to_vocab[valid_examples[i]]
                    top_k = 8 # 取最相似单词的前8个
                    nearest = (-sim[i, :]).argsort()[1:top_k+1]
                    log = 'Nearest to [%s]:' % valid_word
                    for k in range(top_k):
                        close_word = int_to_vocab[nearest[k]]
                        log = '%s %s,' % (log, close_word)
                    print(log)

            iteration += 1
```

在这里注意一下，尽量不要经常去让代码打印验证集相似的词，因为这里会多了一步计算步骤，就是计算相似度，会非常消耗计算资源，计算过程也很慢。所以代码中我设置1000轮打印一次结果。

```
Nearest to six: seven, five, two, four, eight, zero, one, three,
Nearest to world: in, of, international, anmen, borrows, war, ucs, its,
Nearest to can: example, toggle, rearrangement, literals, choices, that, is, chance,
Nearest to be: or, for, by, literals, should, precisely, that, does,
Nearest to called: a, an, is, the, heiress, in, other, lags,
Nearest to zero: two, five, four, six, three, seven, one, eight,
Nearest to one: three, seven, five, four, two, eight, nine, six,
Nearest to b: writer, statesman, drummer, sr, madeleine, ghost, implicates, hiddenstructure,
Nearest to cost: geiger, scaling, minimization, scheduled, joule, printers, py, persecutor,
Nearest to professional: training, disciplines, flexibility, tourists, certified, aquarii, theon, sponsors,
Nearest to defense: knee, verein, bordered, protection, grounding, parcham, cardozo, gallery,
Nearest to lived: loving, grew, charming, erratic, skewed, erectus, aguilera, thought,
Nearest to accepted: began, bosniak, annuity, peacekeeping, maccabean, bylaw, revolved, jeu,
Nearest to gold: silver, medals, copper, tokens, nickel, mined, medal, golden,
Nearest to animals: animal, humans, photojournalist, ais, leopards, organisms, neurotoxins, pest,
Nearest to powers: patagonian, awarding, destroy, tomaso, exercised, branch, bight, avengers,
```

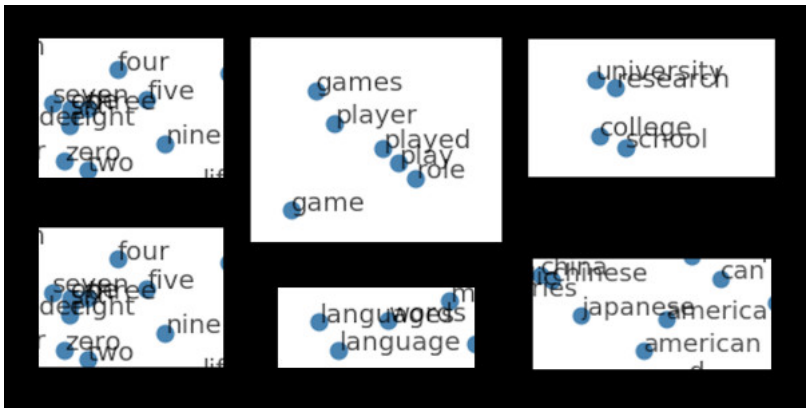
从最后的训练结果来看，模型还是学到了一些常见词的语义，比如one等计数词以及gold之类的金属词，animals中的相似词也相对准确。

为了能够更全面地观察我们训练结果，我们采用sklearn中的TSNE来对高维词向量进行可视化。详见：
http://t.cn/Rofvr7D



上面的图中通过TSNE将高维的词向量按照距离远近显示在二维坐标系中，该图已经在git库中，想看原图的小伙伴去git看~

我们来看一下细节：



上面是显示了整张大图的局部区域，可以看到效果还不错。

关于提升效果的技巧：

- 增大训练样本，语料库越大，模型学习的可学习的信息会越多。
- 增加window size，可以获得更多的上下文信息。
- 增加embedding size可以减少信息的维度损失，但也不宜过大，我一般常用的规模为50-300。

附录：

git代码中还提供了中文的词向量计算代码。同时提供了中文的一个训练语料，语料是我从某招聘网站上爬取的招聘数据，做了分词和去除停用词的操作（可从git获取），但语料规模太小，训练效果并不好。

```
Nearest to [word]: 制造业, 空白, 平者让, 熟练应用, excel, office, 友善, 交予,
Nearest to [ppt]: 网络资源, 操, 透视, 文字处理, 操作, 官网, project, 会计学,
Nearest to [熟悉]: 缺陷, 记录, 不拘泥, 格式, jquery, 所需, 老客户, 法律法规,
Nearest to [java]: 全力, 秉持, 精美, 融入, 安全, 坚守, 参照, 启动,
Nearest to [能力]: 宝, 成交, 计算机, 人事管理, 报警, 估值, 受众, 棒,
Nearest to [逻辑思维]: 自动控制, 登录, 措施, 程序设计, 精准, 有意者, 主人翁, 态度端正,
Nearest to [了解]: 方向和, 求职, 分析方法, 管理学, 下级, 肯吃苦, 高强, 函数,
```

上面是我用模型训练的中文数据，可以看到有一部分语义被挖掘出来，比如word和excel、office很接近，ppt和project、文字处理等，以及逻辑思维与语言表达等，但整体上效果还是很差。一方面是由于语料的规模太小（只有70兆的语料），另一方面是模型也没有去调参。如果有兴趣的同学可以自己试下会不会有更好的效果。

完整代码请见：

http://t.cn/RofPq2p

雷锋网(公众号：雷锋网)相关阅读：

一文详解 Word2vec 之 Skip-Gram 模型（结构篇）

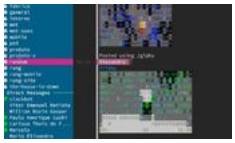
一文详解 Word2vec 之 Skip-Gram 模型（训练篇）

25 行 Python 代码实现人脸检测——OpenCV 技术教程

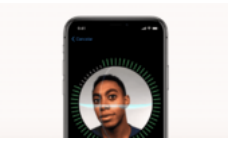
相关文章

Python TensorFlow Skip-Gram

基于 Python 的自动文本提取：抽象法和生成法的比较



开发者必看：8月 Python 热门开放源码



我是如何在 Python 内使用深度学习实现 iPhone X 的

虚谷计划：中国老师发起自己的开源硬件

文章点评：

我有话要说……

同步到新浪微博

提交

最新评论

乐乐努力学习ing 回复: xiaotong 10月22日 15:09

没问题啊 。 官方文档也是这么用的啊https://tensorflow.google.cn/api_docs/python/tf/nn/sampled_softmax_loss

(0) 回复

xiaotong 07月04日 14:58

计算negative sampling下的损失 loss = tf.nn.sampled_softmax_loss(softmax_w, softmax_b, labels, embed, n_sampled, vocab_size) 上句代码有个问题，应该为： loss = tf.nn.sampled_softmax_loss(softmax_w, softmax_b, embed， labels, n_sampled, vocab_size)

(0) 回复

吴恩达正式宣布创业，离职百度92天后成立Deeplearning.ai

本文作者：李秀琴　　2017-06-24 10:02

导语：百度前首席科学家吴恩达宣布创业，离职百度92天后成立Deeplearning.ai。

雷锋网(公众号：雷锋网)6月24日消息 百度前首席科学家吴恩达前几个小时在Twitter上发布一条消息，显示他已建立了一家新的公司：Deeplearning.ai。这则推文只显示该公司的一个标志和一个域名。不过图片中吴恩达还标上了“2017年8月”的信息，这很有可能暗示新公司将于八月正式对外公布。

有媒体发现Deeplearning.ai的域名似乎已注册到百度位于硅谷森尼韦尔的研发机构，也是吴恩达此前工作过的相同的办公室。根据从Wayback Machine上获取的数据，该域名由Instra注册，并于2015-2017年之间被认领。

李秀琴

记者

跟踪Fintech和区块链（微信：cqmm16，备注身份姓名来意，thx）

发私信

当月热门文章

最新文章

吃瓜，生气：arXiv论文里居然有广告？

IEEE Fellow 又增一位华人学者，中科院自动化所王亮研究员当选

云从科技 OCR 新突破：端到端的深度学习文本检测框架 PixelAnchor

李飞飞又有新动向，斯坦福 AI 实验室由 Christopher Manning 接棒

怎样免费拿机械键盘、AI研习社定制双肩包和保温杯？邀请朋友学习「CMU 深度学习课程」即可行！

不一样的论文解读：2018 KDD best paper，Airbnb 的 Embeddings 应用

2

热门搜索

无人机　电动汽车　iOS应用　阿里　中科院　惠普　移动游戏　GoPro　ImageNet　专车

```
DOMAIN INFORMATION
Complete Domain Name.....: deeplearning.ai
Organization Using Domain Name
 Organization Name.....: Baidu
 Street Address.....: 1050 Enterprise Way  Suite 230
 City.....: Sunnyvale
 State.....: CA
 Postal Code.....: 94089
 Country.....: US
Administrative Contact
 NIC Handle (if known).....:
 (I)ndividual (R)ole.....: Owner Andrew
 Name (Last, First).....: Ng
 Organization Name.....: Baidu
 Street Address.....: 1050 Enterprise Way  Suite 230
 City.....: Sunnyvale
 State.....: CA
 Postal Code.....: 94089
 Country.....: US
 Phone Number.....: +1.6507252593
 Fax Number.....:
 E-Mailbox.....: aidomains@instra.com
Technical Contact
 NIC Handle (if known).....:
 (I)ndividual (R)ole.....: Administrator
 Name (Last, First).....: Lentino Tony
 Organization Name.....: Instra Corporation Pty. Ltd.
 Street Address.....: GPO Box 988
 City.....: Melbourne
 State.....: Victoria
 Postal Code.....: 3001
 Country.....: AU
 Phone Number.....: +61.397831800
```

三个月前，吴恩达作为首席科学家的身份离职百度，当时业界纷纷猜测其下一步的去向，很多人认为其非常有可能加入Carol Reiley（吴恩达妻子）所在的Drive.ai公司，投身创业浪潮。不过，据雷锋网此前报道，Carol Reiley在Emtech Digital 2017会场接受采访时否定了这一猜测。

> “我们没有过关于这一话题的谈话，” Reiley说， “他(吴恩达)不会加入Drive.ai。”

结合Reiley的回复，以及Deeplearning.ai这样的域名来看，吴恩达此次创业更有可能是用以开发一种旨在成为关键基础设施以支持人工智能普及的技术。

这在吴恩达离职百度时发表的公开信就可初见端倪，吴恩达在这封信的末尾说道：

> “我将继续致力于用人工智能引领这场重大的社会变革。除了推动大公司使用人工智能，也还有很多创业机会，以及更深入进行人工智能研究的机会。”

当时，Quara上就有人猜测，“吴恩达将会用他丰富的AI经验，推动更多小创业公司使用AI技术。”

并且，在离职后吴恩达接受MIT Technology Review采访时也表示，自己对很多垂直领域的AI应用感兴趣，也表示要花一些时间寻求科技公司转型以外的事情：

> “我决定离开在百度的职位时一切运行地都很好，我想做一些其他的事情……我并不知道自己具体将要做什么，但是我认为AI不仅为百度这样的大公司提供了机遇，也同样为创业公司和基础研究提供机遇。”
>
> “我打算花一些时间，寻求科技公司转型之外的事情。”

> “有很多垂直领域令我感到兴奋，我对医疗领域极其感到兴奋；我对教育领域极其感到兴奋--这些是AI大有可为的主要领域。”

在这周四雷锋网编译的一篇吴恩达张潼接受华尔街日报采访时，吴恩达也重点强调了他对AI+医疗、AI+建筑业，以及美中两国在人工智能技术上的对比的观察。

吴恩达认为，如今AI应用中获利最多的就是在线广告，因为该领域一直都是数字化领域，另外一个就是金融科技领域正在向数字化方向转型。而卫生保健和建筑业在数字化方面稍落后这两个领域。

华尔街日报接着发问吴恩达，什么职业在将来可能已经不存在时，吴恩达也强调表示，AI在医学影像分析上正在取得更好的效果。所以作为放射线技师的职业生涯可能只有五到十年。

作为在中美科技巨头公司均有工作过的这段经历，吴恩达也表达了他对这两个国家在人工智能技术上呈现的差异的看法，吴恩达在采访中表示：

> 美国非常善于创造底层技术，像很多最近的AI技术革新。我认为中国的生态系统在产品推向市场做的非常强，正在以惊人的速度前进。在美国，当你发布一款新产品，总是感觉要打破原先市场分布，再通过市场细分给用户。有太多的市场细分了！
>
> 中国是一个相对同质化的社会。所以，一旦你发现一款适合市场的产品，规模可以扩展的很快。

基于此，我们或有理由猜测，吴恩达成立的Deeplearning.ai这家创业公司将延续其在人工智能、机器学习的研究成果，而主要为医疗、建筑业等领域的创业公司提供人工智能的基础设施技术。

在此之前，吴恩达在任职斯坦福大学副教授时就已做了很多机器学习领域的开拓性工作，且创立了在线教育平台Coursera，在加上在Google Brain、百度的工作经历，这让吴恩达不仅在学术界可称得上是AI领域一位备受尊敬的研究者和传播者，其对工业界而言，也是一位对AI领域的所有形态，包括创业项目、基础技术等具有深刻观察的意见领袖。

距离上次颇具热议的离职事件才刚走过3个月的时间，吴恩达以一如既往的风格在Twitter上率先宣布创业一事。正如吴恩达所说，“人工智能是一次新的'电力革命'，无论你是在哪一个行业工作，AI都有可能改变它。”期待Andrew Ng在一个多月后向大家传达的“AI电力惊喜”。

相关文章：

吴恩达张潼接受WSJ采访：如何让AI像电力一样颠覆世界？

吴恩达妻子Carol Reiley回应：他不会加入Drive.ai

吴恩达离职百度，腾讯任命张潼为AI Lab第一负责人 | AI科技评论周刊

和平分手？你根本不知道吴恩达在百度经历了什么

关于离职百度，吴恩达自己是怎么回应的？

相关文章

吴恩达 创业 Deeplearning.ai 百度

专访吴恩达：AI 将改变所有人类工作，下次寒冬不会

斯坦福发起医疗影像挑战赛，基于吴恩达团队

「我的第一次数据科学家实习经历」

专访 Drive.ai 王弢：吴恩达力挺的首个自动驾驶商业

许家印参观 FF 洛杉矶总部，入股后与贾跃亭首次同

2018国际货币论坛召开 小企业融资问题备受关注

预告 | 大咖Live X 黑芝麻智能科技：ADAS 如何精准感

ACL 2018 首日：8 大 tutorial，深度强化学习最

文章点评：

我有话要说……

同步到新浪微博

提交

热门关键字

热门标签 人工智能 机器人 机器学习 深度学习 金融科技 未来医疗 智能驾驶 自动驾驶 计算机视觉 激光雷达 图像识别 智能音箱 区块链 智能投顾 医学影像 物联网 IoT 微信小程序平台 微信小程序在哪 CES 2017 CES 2016年最值得购买的智能硬件 2016 互联网 小程序 微信朋友圈 抢票软件 智能手机 智能家居 智能手环 智能机器人 智能电视 360智能硬件 智能摄像机 智能硬件产品 智能硬件发展 智能硬件创业 黑客 白帽子 大数据 云计算 新能源汽车 无人驾驶 无人机 大疆 小米无人机 特斯拉 VR游戏 VR电影 VR视频 VR眼镜 VR购物 AR 直播 扫地机器人 医疗机器人 工业机器人 类人机器人 聊天机器人 微信机器人 微信小程序 移动支付 支付宝 P2P 区块链 比特币 风控 高盛 人脸识别 指纹识别 黑科技 谷歌地图 谷歌 IBM 微软 乐视 百度 三星s8 腾讯 三星Note8 小米MIX 小米Note 华为 小米 阿里巴巴 苹果 MacBook Pro iPhone Fac GAIR IROS 双创周 云栖大会 优葩 智能硬件公司 智能硬件 QQ红包 支付宝红包 敬业福 mobike 自行车 华为watch 贵不贵 网盘大战 360儿童手表 app store 支付宝 谷歌 美国大选 实时统计 公众号互选广告 玩具机器人 全国流量卡 共享单车 令计划 乐视 小米mix骨传导 小米单反相机 更多