



上海大学

毕业设计（论文）

订
线

题目：基于 SQuAD 数据集的问题回答模型

学 院：机电工程与自动化学院

专 业：电气工程及其自动化

学 号：15122876

学生姓名：雷清淇

指导教师：周维民

起讫日期：

上海大学学士学位论文

基于 SQuAD 数据集的问题回答模型

姓 名： 雷清淇

导 师： 周维民

学科专业： 自动化

Q&A Model on SQuAD Data Set

Q&A Model on SQuAD Data Set

Candidate:

Supervisor:

Major: Electrical Engineering and Automation

School of Mechatronical Engineering and Automation,

Shanghai University

January, 2018

目 录

摘 要

近数年来，深度神经网络和自然语言处理的结合使 NLP 快速发展，NLP 在翻译、情感分析、文本分类、命名实体识别领域中准确度显著提高。NLP 成为图像识别后另一个深度学习热点，现在 NLP 在应用方向发展很好，在智能音箱、智能客服、文本审查、机器翻译方面有实际的产品。

目前在准确率、训练复杂度这两方面均有不错表现的目标检测方法是采用 Transformer 的 BERT，它可以作为一种可以预训练的自注意力 NLP 模型。该方法可以只需要一次预训练，接下来所有 NLP 任务既可以使用有限的训练样本微调，也可以直接使用。BERT 的出现使得人们可以使用在 NLP 领域表现比 RNN 更加优秀的 Transformer。本课题针对斯坦福大学的 SQuAD 问答数据集，使用 BERT 模型训练并测试模型的准确度。

本次课题的数据训练是在 Ubuntu16.04 操作系统上进行的，因为在 Linux 上 Tensorflow 更加稳定。在显卡驱动、cuda、Tensorflow 安装的基础上运用 Bert 提供的源代码，我训练了一种基于 Transformer、自注意力机制的问题回答模型。最后比较新模型新算法的准确度和训练复杂度后,我分析了未来 NLP 发展的方向。

关键词：自然语言处理；Transformer；BERT；SQuAD

ABSTRACT

In recent years, the combination of deep neural networks and natural language processing has enabled NLP to develop rapidly. NLP has significantly improved accuracy in the fields of translation, sentiment analysis, text classification, and named entity recognition. NLP has become another deep learning hotspot after image recognition. Now NLP has developed very well in the industry, and has a good product in smart speakers, intelligent customer service, text review, and machine translation.

At present, the word embedding generation method with good performance in both accuracy and training complexity is BERT, which can be used as a pre-trained self-attention NLP model. This method only requires one pre-training, and then all NLP tasks can be either fine-tuned with limited training samples or carried on directly. The emergence of BERT allows people to use Transformer that performs better than RNN in the NLP field. This topic is aimed at Stanford University's SQuAD Q&A data set, using the BERT model to train and test the accuracy of the model.

The data training for this topic was carried out on the Ubuntu 16.04 operating system because Tensorflow is more stable on Linux. On the basis of the graphics card driver, cuda, Tensorflow, using the source code provided by Bert, I trained a problem answer model based on Transformer and self-attention. Finally, after comparing the accuracy and training complexity of the new model and new algorithm, I analyzed the future direction of NLP development.

Keywords: Face Detection; License System; Convolutional Neural Network; Faster R-CNN; Django Framework

第1章 绪论

1.1 课题背景及意义

这个课题是美国斯坦福大学本科课程 CS224n 自然语言处理的课程设计，它要使用循环神经网络（RNN）设计一个模型来训练 SQuAD 数据集，并试图在测试集上获得超过人类表现的水平。研究这个课题，对于如何让电脑理解文章并对文章相关的问题做出回答有重要意义。

1.2 课题研究现状

课题研究领域发展速度很快。深度学习是当下的一大热点，自然语言处理和深度学习的结合，使得当下自然语言处理的研究达到了近数十年的新高度。目前主要使用 RNN（循环神经网络）设计模型来训练数据，google 在 2018 年 10 月开源了 BERT(Bidirectional Encoder Representations from Transformers)，这使得自然语言处理的发展达到近数年的新高度。

未来一段时间自然语言处理发展方向可能是 BERT 的应用更加广泛，实用性更强。各个深度学习框架（Pytorch, Tensorflow, Keras）开始支持 BERT。

1.3 论文主要研究内容

SQuAD 数据集是美国斯坦福大学在维基百科上摘录数万个段落，并利用众包方式，进行了给定文章，提问题并给答案的人工标注。他们将这两万多个段落给不同人，要求对每个段落提五个问题。

这个数据集的评测标准有两个，第一：F1，第二：EM。EM 是完全匹配的缩写，必须机器给出的和人给出的一样才算正确。而 F1 是将答案的短语切成词，和人的答案一起算 recall, Precision 和 F1，即如果你 match 了一些词但不全对，

仍然算分。

要达到的目标是学习深度学习相关知识，编写代码，最后运行 SQuAD2.0 测试集 F1 得分达到 70 分。

本文中使用的模型是结合 Google 最新理论和实践成果 BERT(Bidirectional Encoder Representations from Transformers)来设计问题回答模型。使用自然语言处理方向上新的数据集，SQuAD 是这个方向上最具影响力的数据集之一。本次课题研究中，会直接学习并采用 Google 的现有研究成果，这样可以节约人力物力，并使模型的效果接近现在已有的最高水平。

全文共分五章，安排如下：

第一章：绪论。毕业设计课题来源、背景及研究意义，人脸检测的研究现状，此次课题研究内容的总结。

第二章：神经网络和监督学习的关系、深度学习为什么发展这么快。逻辑回归和神经网络的基本算法：线性变换、sigmoid 非线性变换，ReLU 非线性变换、损失函数、代价函数、向前传播、向后传播、梯度下降、并给出使用 Python 中 Numpy 搭建的神经网络模型。

第三章：神经网络的各种优化算法。主要包括 Mini-batch，避免过拟合 L2 正则化、Dropout，避免梯度消失和梯度爆炸的参数初始化。常用的三种梯度下降算法：Momentum、RmSprop、Adam 的公式推导和理解。

第四章：卷积神经网络的基本概念的介绍和公式的推导。

第五章：近数年来，循环神经网络的发展历史，各个时期的热门算法和模型的优点和不足。Word embedding、Word2Vec、GRU、LSTM、Transformer、BERT 相关公式的推导，模型的架构。

第六章：自然语言处理方面的模型，和各个模型的优点和不足

第七章：使用 BERT 训练并测试 SQuAD2.0 数据集，对结果评估。

第八章：总结与展望。对毕业设计中完成的工作进行了总结，并对以后要做的事情进行了展望。

第2章 神经网络

本章内容主要是深度学习基本算法，给出了逻辑回归（Logistic regression），深度神经网络的基本推导过程、各种激活函数的介绍，以及一个可以进行猫与非猫识别的深度神经网络实现代码。

2.1 监督学习和神经网络

目前为止，由神经网络模型创造的价值基本上都是基于监督式学习（Supervised Learning）的。监督式学习与非监督式学习本质区别就是是否已知训练样本的输出 y 。在实际应用中，机器学习解决的大部分问题都属于监督式学习，神经网络模型也大都属于监督式学习。

数据类型一般分为两种：结构化数据和非结构化数据。结构化数据通常指的是有实际意义的数据。例如某一商品房的中的大小，价格，卧室数量等。这些数据都具有实际的物理意义，比较容易理解。而非结构化数据通常指的是比较抽象的数据，例如音频，图像或者文字段落。以前，计算机对于非结构化数据比较难以处理，而人类对非结构化数据却能够处理的比较好，例如我们第一眼很容易就识别出一张图片里是否有猫，但对于计算机来说并不那么简单。现在，值得庆幸的是，由于深度学习和神经网络的发展，计算机在处理非结构化数据方面效果越来越好，甚至在某些方面优于人类。总的来说，神经网络与深度学习无论对结构化数据还是非结构化数据都能处理得越来越好，并逐渐创造出巨大的实用价值。

2.2 为什么现在深度学习的发展日新月异

深度学习和神经网络背后的技术思想已经出现数十年了，那么为什么直到现在才开始发挥作用呢？

Scale drives deep learning progress

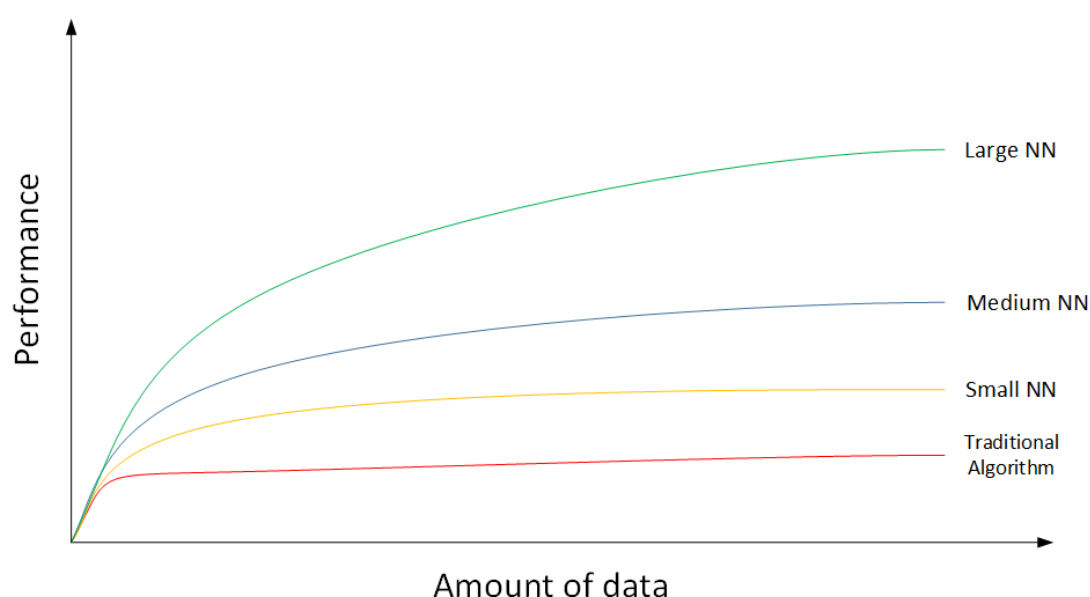


图 2- 2 模型效果比较图

上图共有 4 条曲线。红色曲线代表了传统机器学习算法的表现，例如是支持向量机，逻辑回归，决策树等。当数据量比较小的时候，传统学习模型的表现是比较好的。但是当数据量很大的时候，性能基本不会变得更好。黄色曲线代表了规模较小的神经网络模型（Small NN）。它在数据量较大时候的性能优于传统的机器学习算法。蓝色曲线代表了规模中等的神经网络模型（Media NN），它在在数据量更大的时候的表现比 Small NN 更好。绿色曲线代表更大规模的神经网络（Large NN），即深度学习模型。在数据量很大的时候，深度学习模型的表现仍然是最好的，而且基本上保持了较快上升的趋势。值得一提的是，近些年来，由于数字计算机的普及，人类进入了大数据时代，每时每分，互联网上的数据是海量的、庞大的。如何对大数据建立稳健准确的学习模型变得尤为重要。传统机器学习算法在数据量较大的时候，性能一般，很难再有提升。然而，深度学习模型由于网络复杂，对大数据的处理和分析非常有效。所以，近些年来，在处理海量数据和建立复杂准确的学习模型方面，深度学习有着非常不错的表现。然而，在数据量不大的时候，例如上图中左边区域，深度学习模型不一定优于传统机器学习算法，性能差异可能并不大。

所以说，现在深度学习如此强大的原因归结为三个因素：Data, Computation, Algorithms。

其中，数据量的几何级数增加，加上 GPU 算力突飞猛进、计算机运算能力的大大提升，使得深度学习能够应用得更加广泛。另外，算法上的创新和改进让深度学习的性能和速度也大大提升。举个算法改进的例子，之前神经网络神经元的激活函数是 Sigmoid 函数，后来改成了 ReLU 函数，在实际应用中采用 ReLU 函数确实要比 Sigmoid 函数快很多。

2.3 逻辑回归

逻辑回归模型一般用来解决二分类，二分类就是输出 y 只有 $\{0,1\}$ 两个离散值（也有 $\{-1,1\}$ 的情况）。我们以一个图像识别问题为例，判断图片中是否有猫存在，0 代表没有猫，1 代表有猫。

上图所示，cat 图片的尺寸为 $(64,64,3)$ 。将图片转化为一维的特征向量 x 。则转化后的输入特征向量维度为 $(12288,1)$ 。此特征向量是列向量 x ，维度一般记为 n_x 。训练样本共有 m 张图片，那么整个训练样本 x 组成了矩阵，维度是 (n_x, m) 。

在逻辑回归中，我们会引入参数 w 和 b 。权重 w 的维度是 $(n_x, 1)$ ， b 是一个常数项（在 python 程序中，矩阵减去一个常数相当于所有矩阵元素减去那个常数）。逻辑回归中有线性变换和非线性变换，线性变换为：

$$z = w^T x + b, \quad (\text{矩阵维度: } (1, m) = (n_x, 1)^T (n_x, m) + (1, m))$$

线性输出区间为整个实数范围 $(-\infty, +\infty)$ ，而逻辑回归要求输出范围在 $[0,1]$ 之间，可以引入非线性函数 Sigmoid，此变换矩阵维度不变。

$$\text{Sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = a = \text{Sigmoid}(w^T x + b) = \sigma(w^T x + b) = \sigma(z), \quad (\hat{y} \text{ 矩阵维度 } (1, m))$$

接下来，我们引入损失函数（Loss Function），它的目的就是要衡量预测一个样本输出 \hat{y} 与其真实结果 y 的接近程度。因为一个样本的输出 y 是一个自然数，损失函数计算结果也是一个自然数，

$$L(\hat{y}^{(i)}, y^{(i)}) = -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

这个 Loss Function 是针对单个样本的。但是我们在训练时，不会一次只训练一个样本，一般我们把 m 个样本的输入矩阵叠加到一个矩阵 x 中，对于 m 个样本，可以定义代价函数（Cost Function），这里使用右上标表示某个样本，输出是一个自然数：

$$\begin{aligned} J(w, b) &= \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) \\ &= -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \end{aligned}$$

代价函数用来评价模型的好坏，其值越小说明模型和当前参数可以更好地预测。代价函数目标是计算出使用当前 w 和 b 时， m 个训练样本的输出与真实值相差多大。我们的最小化 Cost Function，让 Cost Function 尽可能地接近于零。怎么最小化，我们知道一个函数在一个点的导数代表着函数在当前点上函数值上升或者下降的快慢，而接下来的梯度下降就是不断使用 Cost Function 的导数来最小化 Cost Function。

2.4 梯度下降

梯度下降算法是先随机选择一组参数 w 和 b 值，然后每次迭代的过程中分别沿着 w 和 b 的梯度（偏导数）的反方向前进一小步，不断修正 w 和 b 。每次迭代更新 w 和 b 后，都能让 $J(w, b)$ 更接近全局最小值。 w 和 b 的修正表达式为：

$$\begin{aligned} w &:= w - \alpha \frac{\partial J(w, b)}{\partial w} \\ b &:= b - \alpha \frac{\partial J(w, b)}{\partial b} \end{aligned}$$

接下来梯度下降推导使用 dw 代替损失函数对 w 的偏导数，并且 dw 矩阵维度和 w 矩阵维度相同，以此类推。

2.4.1 单个样本的梯度下降

对于 1 样本，需要求损失函数（Loss Function）关于 w 和 b 的偏导数：我们使用 da 表示损失函数对 a 的偏导数，以此类推：

$$da = \frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{\partial a}{\partial z} = \text{Sigmoid}(z)' = \frac{e^{-x}}{(1 + e^{-x})^2} = \text{Sigmoid}(z)(1 - \text{Sigmoid}(z))$$

$$dz = \frac{\partial L}{\partial a} * \frac{\partial a}{\partial z} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) * a(1-a) = a - y$$

可以看到，损失函数对线性变换的结果 z 的偏导数是预测结果 a 减去真实值 y ，知道了 dz ，可以很容易对 w 和 b 求导：

$$dw = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial w} = x * dz = x(a - y)$$

$$db = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial b} = 1 * dz = a - y$$

梯度下降就是 w 和 b 减去学习率和 w 与 b 的偏导数，从而让损失函数计算出来的值变小，让预测值接近于真实值：

$$w = w - \alpha dw$$

$$b = b - \alpha db$$

2.4.2 m 个样本的梯度下降

因为图片输入是 m 个样本一起输入，对于 m 个样本，需要对每一个样本求损失函数（Loss Function），我们使用右上标并加括号表示某一个样本， $x^{(i)}$ 表示第 i 个样本的输入数据，以此类推：

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})$$

关于 w 和 b 的偏导数并求平均：

$$dw = \frac{1}{m} x dz^T$$

$$db = \frac{1}{m} \sum_{i=1}^m (a^{(i)} - y^{(i)})$$

因为 b 是一个自然数，所以 db 也要是一个数字，这里把维度为 $(1, m)$ 的 dz 第一行的元素相加求平均得到 db 。而 w 维度是 $(n_x, 1)$ ，需要 $x dz^T, (n_x, m)(1, m)^T$ 。

经过一次迭代后，梯度下降，对 w 和 b 进行更新：

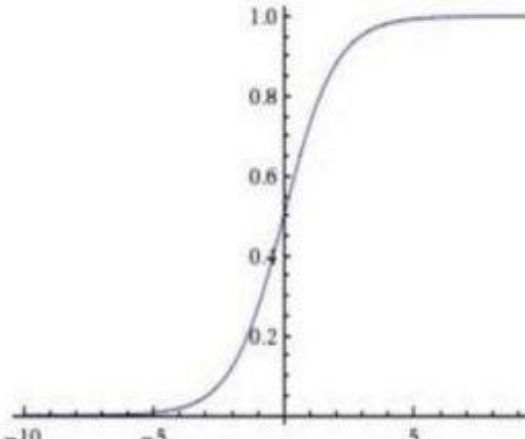
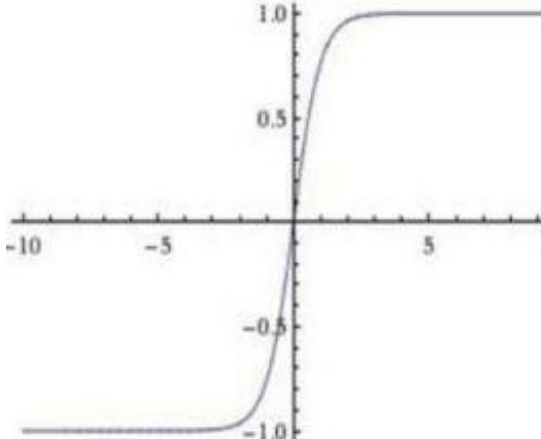
$$w = w - \alpha dw$$

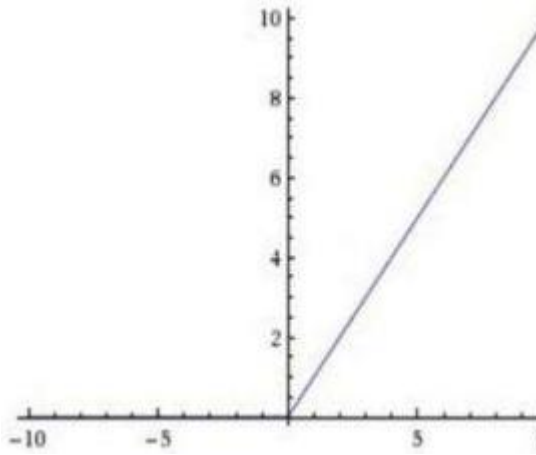
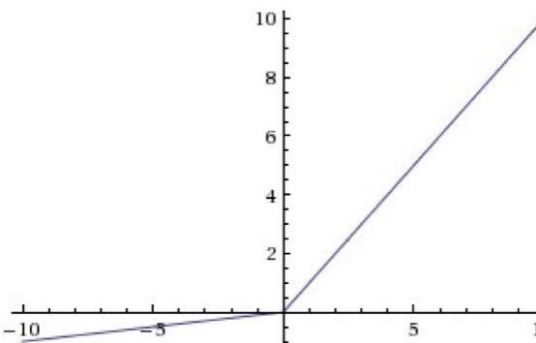
$$b = b - \alpha db$$

这样经过 n 次迭代，整个深度学习算法中学习部分就完成了，深度学习就是通过梯度下降不断对网络中参数更新，从而使代价函数计算出的数值不断降低让预测值接近于真实值。

2.5 激活函数

神经网络的隐藏层和输出层都需要激活函数，不同激活函数有各自用处。

名称	函数	图像	导数
Sigmoid	$\frac{1}{1 + e^{-z}}$		$a(1 - a)$
Tanh	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$		$1 - a^2$

ReLU	$\max(0, z)$		$\begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$
Leaky ReLU	$\max(0.01z, z)$		$\begin{cases} 1, & z \geq 0 \\ 0.01, & z < 0 \end{cases}$

在实际使用中，sigmoid 函数使用在二分类的输出层，sigmoid 函数和 tanh 函数的缺陷是当 z 很大或者很小的时候，激活函数的斜率很小，因此，梯度下降算法会运行的比较慢。同时，由于需要指数运算，计算比 Relu 更费时间。

ReLU 避免了斜率很小的这个缺陷，在 2010 年左右很多深度学习研究者还认为 sigmoid 函数比 ReLU 效果更好，但是随着近几年计算速度（主要是 GPU 的计算速度）的提高，研究者在进行了大量数据的训练后发现神经网络使用 ReLU，计算速度更快，效果使用 sigmoid 相似。对于隐藏层，选择 ReLU 作为激活函数能够保证 z 大于零时梯度始终为 1，从而提高神经网络梯度下降算法运算速度。当 z 小于零，存在梯度为 0 的缺点，但在实际应用中，这个缺点影响不大。出现了 Leaky ReLU 函数，能够保证 z 小于零是梯度不为 0。

实际应用中，隐藏层通常会使用 ReLU 函数或者 Leaky ReLU 函数，其实，具体选择哪个函数作为激活函数没有一个固定的准确的答案，应该要根据具体实际问题进行验证。

2.6 神经网络

逻辑回归可以理解为一个神经元，神经网络包含多个神经元。如下图所示，分别列举了逻辑回归、1 个隐藏层的神经网络、2 个隐藏层的神经网络和 5 个隐藏层的神经网络它们的模型结构。深层神经网络其实就是包含更多的隐藏层浅层神经网络。

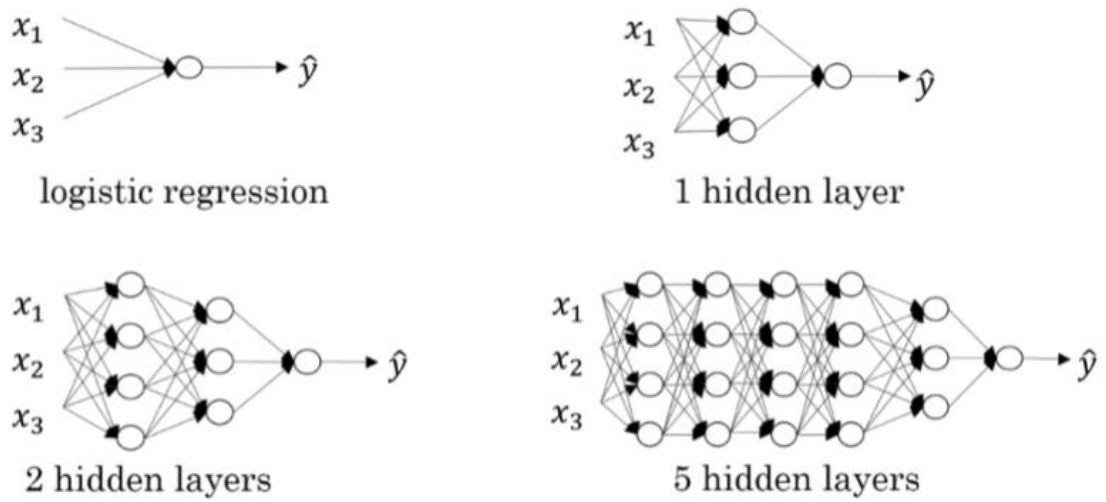


图 2-3 逻辑回归和神经网络

神经网络的层数只参考隐藏层个数和输出层，图 2-3 中，逻辑回归又叫 1 层神经网络，10 层隐藏层，2 层输出层的神经网络叫做 12 层神经网络，以此类推。总层数用 L 表示，输入层是第 0 层，输出层是第 L 层。神经网络的形状使用一个数组 n 来表示， $n^{[3]} = 5$ 表示第三层有 5 个神经元。对于神经网络，用大写 $W^{[l]}$ 和 $b^{[l]}$ 来表示第 l 层参数，第 0 层（输入层）没有参数 W 和 b 。对于 m 个训练样本，各层参数维度：

输入 $A^{[l-1]}$: $(n^{[l-1]}, m)$

参数 $W^{[l]}$: $(n^{[l]}, n^{[l-1]})$

参数 $b^{[l]}$: $(n^{[l]}, 1)$

输出 $A^{[l]}$: $(n^{[l]}, m)$

在 Python 程序中，由于 Numpy 的广播性质， $b^{[l]}$ 维度会被当成 $(n^{[l]}, m)$ 矩阵进

行运算， $dW^{[l]}$ 和 $db^{[l]}$ 的维度和 $W^{[l]}$ 和 $b^{[l]}$ 维度相同。参数 W 不需要想逻辑回归中的 w 转秩。

2.6.1 神经网络中向前传播和向后传播

神经网络中，使用 $Z^{[l]}$ 表示第 l 层线性变换结果， $A^{[l]}$ 表示第 l 层激活函数输出结果。向前传播中第 l 层输入是第 $l-1$ 层的输出，利用输入的 $A^{[l]}$ 计算出 $W^{[l]}$ 和 $b^{[l]}$ 。向后传播中第 l 层输入是第 $l+1$ 层的输出，利用输入的 $da^{[l]}$ 计算出 $dW^{[l]}$ 和 $db^{[l]}$ 。

向前传播比较简单，最后一层激活函数使用 sigmoid 函数，其他隐藏层都是用 ReLU 函数：

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

\vdots

$$A^{[L]} = g^{[L]}(Z^{[L]}) = \hat{Y}$$

反向传播如图 2-4 所示利用上一层算出的 $da^{[L]}$ ，先计算这一层的 $dZ^{[L-1]}$ ：

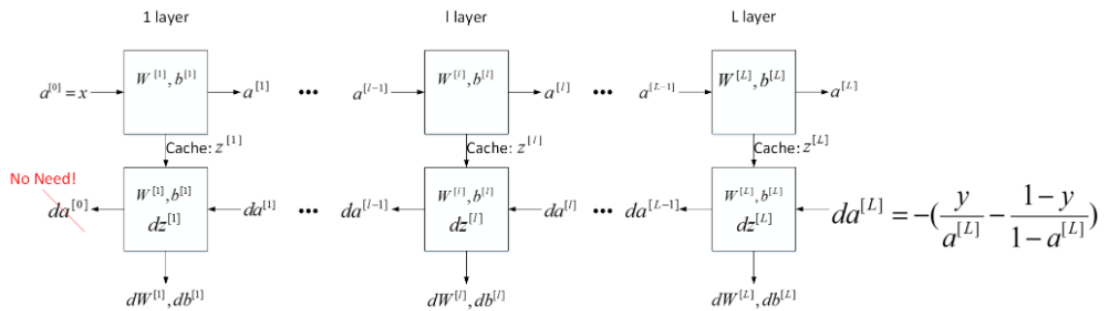


图 2-4 向前传播和向后传播

$$dZ^{[L-1]} = dA^{[L]} g'^{[L]}(Z^{[L-1]})$$

在使用 $dZ^{[L-1]}$ ，计算出 $dW^{[L-1]}$ ， $db^{[L-1]}$ ， $dA^{[L-1]}$ ，不断向后计算。完整代码如下：

$$da^{[L]} = -\left(\frac{y}{a^{[L]}} - \frac{1-y}{1-a^{[L]}}\right), \quad \text{损失函数的导数}$$

$$dZ^{[L]} = A^{[L]} - Y, \quad \text{最后一层损失函数对线性输出的导数}$$

$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{[L-1]T}, \quad \text{最后一层损失函数对 } W \text{ 的偏导}$$

$$db^{[L]} = \frac{1}{m} \text{np.sum}(dZ^{[L]}, \text{axis} = 1, \text{keepdims} = \text{True}), \quad \text{最后一层损失函数对 } b \text{ 的偏导}$$

$$dA^{[L]} = W^{[L]T} dZ^{[L]},$$

最后一层损失函数对输入 $A^{[L-1]}$ （倒数第二层的输出）的偏导

$$dZ^{[L-1]} = dA^{[L]} g'^{[L-1]}(Z^{[L-1]}) = W^{[L]T} dZ^{[L]} g'^{[L-1]}(Z^{[L-1]}),$$

在倒数第二层使用最后一层计算出的 $dA^{[L]}$

⋮

$$dZ^{[1]} = dA^{[2]} g'^{[1]}(Z^{[1]}) = W^{[2]T} dZ^{[2]} g'^{[1]}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]T}$$

$$db^{[0]} = \frac{1}{m} \text{np.sum}(dZ^{[0]}, \text{axis} = 1, \text{keepdims} = \text{True})$$

2.7 深度学习框架

使用开源框架，自己可以不必实现神经网络的每一个细节，只需要著重于网络的设计和训练。使用 Keras，一层神经网络或者所有的反向传播只需要一行代码。训练速度可能比自己实现的更快，因为框架底层都是使用 C/C++ 来编写的，并不是 Python，这自然比纯 Python 的神经网络运行速度快。框架的价值也可以从耗费资金来看，Tensorflow 是 Google 开发的，应该有超过一百个软件工程师和算法工程师从事其的开发工作，Google 软件工程师薪水普遍在数十万美元一

年，也就是说这几年 Google 投入到 Tensorflow 上的钱就要用亿美元来计算，耗资之大从侧面表现了这些开源框架的强大。

2.7.1 Tensorflow

2015 年 11 月 10 日,Google 宣布推出全新的机器学习开源工具 TensorFlow。TensorFlow 最初是由 Google 机器智能研究部门的 Google Brain 团队开发,基于 Google 2011 年开发的深度学习基础架构 DistBelief 构建起来的。Tensorflow 是现在最流行的深度学习框架,它被大部分公司使用,获得了极大的成功,但它也有缺点,如下:

过于复杂的系统设计, Tensorflow 在 GitHub 代码仓库的代码量超过 100 万行。

接口设计过于晦涩难懂。在设计 TensorFlow 时,创造了图、会话、命名空间、Placeholder 等诸多抽象概念,对普通用户来说难以理解。

直接使用 Tensorflow 生产力低下, Google 官方和其他开发者尝试构建更容易使用的接口,包括 Keras、Sonnet、TFLearn、TensorLayer、Slim、Fold、PrettyLayer 等数不胜数的第三方框架每隔几个月就会在新闻中出现一次。

凭借着 Google 强大的推广能力, TensorFlow 已经成为当今最炙手可热的深度学习框架。但在 2018 年和 2019 年上半年, Facebook 开源的 PyTorch 大受欢迎,渐渐成为和 Tensorflow 一样流行、成熟的深度学习框架。

2.7.2 Pytorch

2017 年 1 月, Facebook 人工智能研究院团队在 GitHub 上开源了 PyTorch, PyTorch 随之成为 Tensorflow 的主要竞争对手。PyTorch 的设计追求最少的封装, 尽量避免重复造轮子。PyTorch 的设计遵循 `tensor`→`variable`(`autograd`)→`nn.Module` 三个由低到高的抽象层次, 分别代表高维数组(张量)、自动求导(变量)和神经网络(层/模块), 而且这三个抽象之间联系紧密, 可以同时进行修改和操作。PyTorch 是当前难得的简洁优雅且高效快速的框架。它的源码只有 TensorFlow 的

十分之一左右，更少的抽象、更直观的设计使得 PyTorch 的源码十分易于阅读。它的灵活性不以速度为代价，其速度与 Tensorflow 相差不大。

2.7.3 Keras

Keras 应该是深度学习框架之中最容易上手的一个。Keras 并不能称为一个深度学习框架，它更像一个深度学习接口，它构建于第三方框架之上。学习 Keras 十分容易，搭建一层神经网络只需要一行代码。但是在使用 Keras 的大多数时间里，用户主要是在调用接口，很难真正学习到深度学习的内容。

但是对于已经熟悉深度学习基本理论知识的人，Keras 是开发速度最快的框架。Keras 是 Tensorflow 的一部分，安装好 Tensorflow 后，就可以使用 Keras 了。

2.10 使用 Numpy 实现深度神经网络

下面的代码是我用 Python 中 Numpy 库，实现这一章中的算法。

第3章 神经网络的优化

本章给出了针对普通深度神经网络算法的优化，主要包括训练集、验证集、测试集的分配，防止过拟合和欠拟合的方法，防止梯度爆炸和梯度消失的方法，以及优化的梯度下降算法。经过优化的模型更加健壮，梯度下降速度更快。这些优化，是深度学习不可缺少的一部分。

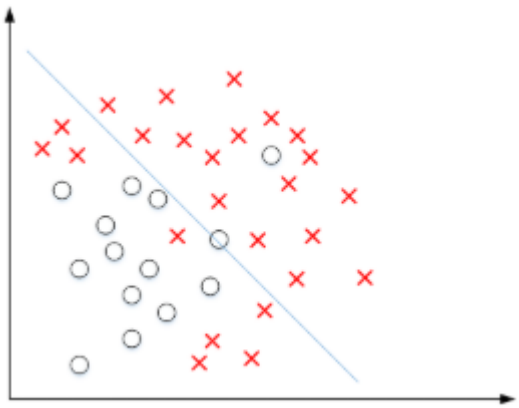
3.1 训练集/验证集/测试集

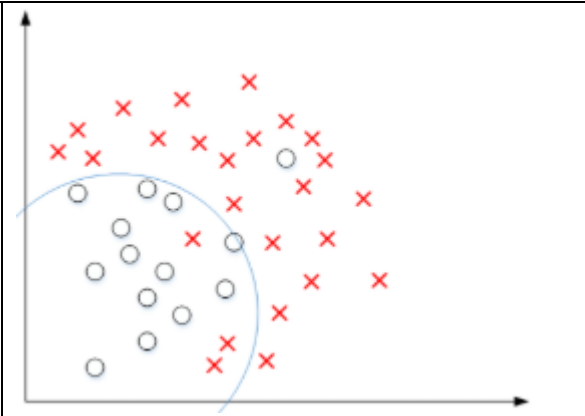
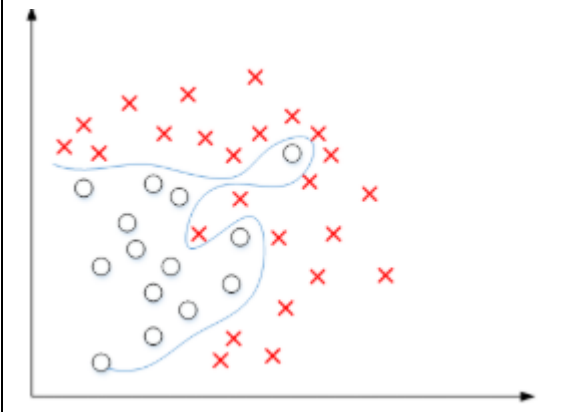
一般地，我们将所有的样本数据分成三个部分：Train/Dev/Test sets。Train sets 用来训练你的算法模型；Dev sets 用来验证不同算法的表现情况，从中选择最好的算法模型；Test sets 用来测试最好算法的实际表现。之前人们通常设置 Train sets 和 Test sets 的数量比例为 70%和 30%。如果有 Dev sets，则设置比例为 60%、20%、20%，分别对应 Train/Dev/Test sets。这种比例分配在样本数量不是很大的

情况下，例如 100,1000,10000，是比较科学的。但是如果数据量很大的时候，例如 100 万，这种比例分配就不太合适了。科学的做法是要将 Dev sets 和 Test sets 的比例设置得很低。因为 Dev sets 的目标是用来比较验证不同算法的优劣，从而选择更好的算法模型就行了。因此，通常不需要所有样本的 20%这么多的数据来进行验证。对于 100 万的样本，往往只需要 10000 个样本来做验证就够了。对于大数据样本，Train/Dev/Test sets 的比例通常可以设置为 98%/1%/1%，或者 99%/0.5%/0.5%。

3.2 过拟合和欠拟合

数据和模型的拟合情况是很重要的一个问题。判断欠拟合和过拟合主要通过训练样本和测试样本的损失或者正确率。减少欠拟合的主要方法是增加神经网络的复杂程度，训练时间延长，而减少过拟合主要有降低神经网络复杂度、正则化 (Regularization)、Dropout，增加训练样本数。

	图例	训练时误差可能的情况
欠拟合		Train set error: 15% Dev set error: 16%

拟合		Train set error: 15% Dev set error: 10%
过拟合		Train set error: 1% Dev set error: 11%

3.2.1 正则化

如果出现了过拟合，可以使用正则化(Regularization)来解决。虽软扩大训练样本数量也是一种方法，但是通常获得更多的训练样本比较困难。

数据和模型的拟合情况是很重要的一个问题。判断欠拟合和过拟合主要通过训练样本和测试样本的损失或者正确率。减少欠拟合的主要方法是增加神经网络的复杂程度，训练时间延长，而减少过拟合主要有降低神经网络复杂度、正则化 Regularization、Dropout，增加训练样本数。对成本函数(Cost function)加入 L2 正则化：

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$$

$$\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$$

$\|w\|_2^2$ 是计算所有元素平方和再开放。由于加入了正则化项，梯度下降算法中的 dw 计算表达式需要做如下修改：

$$dw^{[l]} = dw_{before}^{[l]} + \frac{\lambda}{m} w^{[l]}$$

$$w^{[l]} = w^{[l]} - \alpha dw^{[l]}$$

L2 Regularization 也被称作 weight decay。这是因为，由于加上了正则项， $dw^{[l]}$ 有个增量，在更新 $w^{[l]}$ 的时候，会多减去这个增量，使得 $w^{[l]}$ 比没有正则项的值要小一些。不断迭代更新，不断地减小。 $w^{[l]}$ 近似为零，意味着该神经网络模型中的某些神经元实际的作用很小，可以忽略。从效果上来看，其实是将某些神经元给忽略掉了。这样原本过于复杂的神经网络模型就变得不那么复杂了，而变得非常简单化了。这样过拟合的情况就被减少了。

3.2.2 Dropout

除了 L2 Regularization，Dropout 也可以防止过拟合。Dropout 是指在深度学习网络的训练过程中，对于每层的神经元，按照一定的概率将其暂时从网络中丢弃。也就是说，每次训练时，每一层都有部分神经元不工作，起到简化复杂网络模型的效果，从而避免发生过拟合。Dropout 有不同的实现方法，接下来介绍一种常用的方法：Inverted dropout。假设对于第 1 层神经元，设定保留神经元比例概率 $keep_prob=0.8$ ，即该层有 20% 的神经元停止工作。

对于 m 个样本，单次迭代训练时，随机删除掉隐藏层一定数量的神经元；然后，在删除后的剩下的神经元上正向和反向更新权重 w 和常数项 b ；接着，下一次迭代中，再恢复之前删除的神经元，重新随机删除一定数量的神经元，进行正向和反向更新 w 和 b 。不断重复上述过程，直至迭代训练完成。从效果上来说，与 L2 regularization 是类似的，都是对权重 w 进行“惩罚”，减小了 w 的值。

值得注意的是，使用 dropout 训练结束后，在测试和实际应用模型时，不需要进行 dropout 和随机删减神经元，所有的神经元都在工作。

使用 dropout 的时候，可以通过绘制 cost function 来进行 debug，看看 dropout 是否正确执行。一般做法是，将所有层的 $keep_prob$ 全设置为 1，再绘制 cost

function，即涵盖所有神经元，看 J 是否单调下降。下一次迭代训练时，再将 `keep_prob` 设置为其它值。

3.2.3 数据增加和早期停止

除了 L2 regularization 和 dropout regularization，还有其他减少过拟合的方法。

一种方法是增加训练样本数量。但是通常成本较高，难以获得额外的训练样本。但是，我们可以对已有的训练样本进行一些处理来“制造”出更多的样本，称为 data augmentation。

还有另外一种防止过拟合的方法：early stopping。一个神经网络模型随着迭代训练次数增加，train set error 一般是单调减小的，而 dev set error 先减小，之后又增大。因此，迭代训练次数不是越多越好，可以通过 train set error 和 dev set error 随着迭代次数的变化趋势，选择合适的迭代次数，即 early stopping。

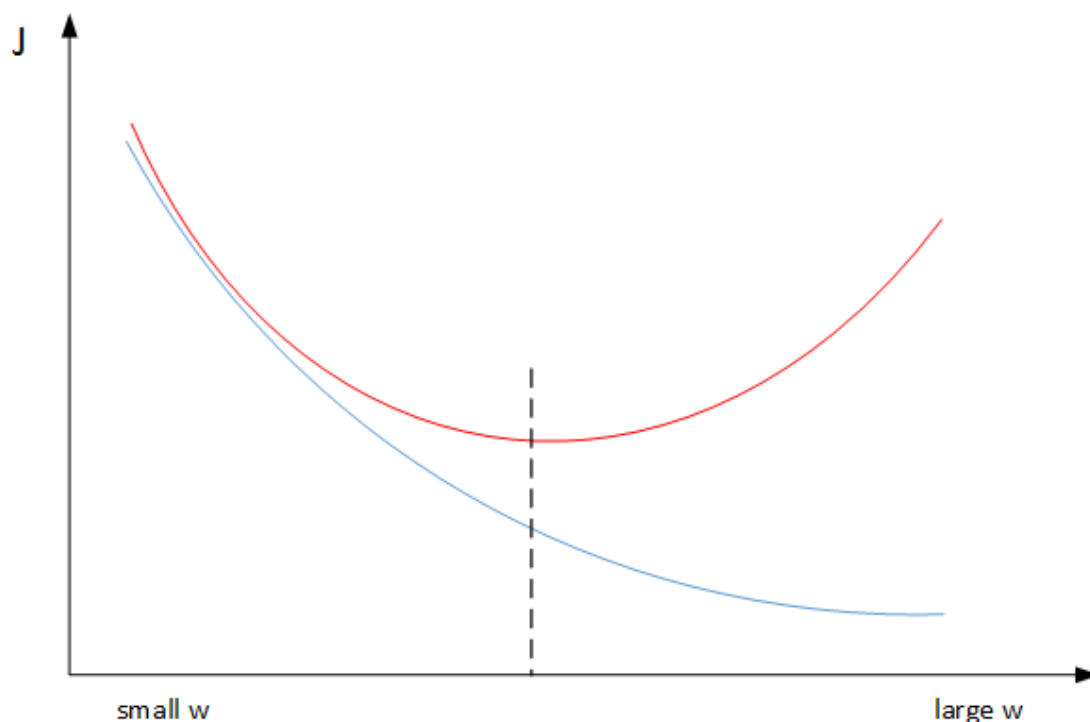


图 3-1 过拟合训练和验证集的 error

然而，Early stopping 有其自身缺点。通常来说，机器学习训练模型有两个目标：一是优化 cost function，尽量减小 J ；二是防止过拟合。这两个目标彼此对立的，即减小 J 的同时可能会造成过拟合，反之亦然。我们把这二者之间的关系称

为正交化 orthogonalization。该节课开始部分就讲过，在深度学习中，我们可以同时减小 Bias 和 Variance，构建最佳神经网络模型。但是，Early stopping 的做法通过减少得带训练次数来防止过拟合，这样 J 就不会足够小。也就是说，early stopping 将上述两个目标融合在一起，同时优化，但可能没有“分而治之”的效果好。

与 early stopping 相比，L2 regularization 可以实现“分而治之”的效果：迭代训练足够多，减小 J，而且也能有效防止过拟合。而 L2 regularization 的缺点之一是最优的正则化参数 λ 的选择比较复杂。对这一点来说，early stopping 比较简单。总的来说，L2 regularization 更加常用一些。

3.3 梯度爆炸和梯度消失

在神经网络尤其是深度神经网络中存在可能存在这样一个问题：梯度消失和梯度爆炸。意思是当训练一个 层数非常多的神经网络时，计算得到的梯度可能非常小或非常大，甚至是指数级别的减小或增大。这样会让训练过程变得非常困难。

一般，在隐藏层，我们使用 ReLU 函数，ReLU 函数在输入为正数部分输出为线性，即 $g(Z) = Z$ 。那么我们忽略各层常数项 b 的影响，令 b 全部为零。这是前面推到出来的公式：

$$dZ^{[1]} = dA^{[2]} g'^{[1]}(Z^{[1]}) = W^{[2]T} dZ^{[2]} g'^{[1]}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]T}$$

如果各层 $W^{[l]}$ 的元素都稍大于 1，例如 1.5，那么 dZ 就会很大。A 是当前层的输入，上一层的输出，大数值的 W 也会让 A 变大，这就导致 dW 的变大，从而造成梯度爆炸。

也就是说，如果各层权重 W 都大于 1 或者都小于 1，那么各层激活函数的输出将随着层数 l 的增加，呈指数型增大或减小。当层数很大时，出现数值爆炸或消失。同样，这种情况也会引起梯度呈现同样的指数型增大或减小的变化。L 非

常大时，例如 $L=150$ ，则梯度会非常大或非常小，引起每次更新的步进长度过大或者过小，这让训练过程十分困难。

3.4 参数初始化

我们可以使用一些方法来避免梯度消失和梯度爆炸，方法是对权重 w 进行一些初始化处理。对于 ReLU 函数，将随机生成的 W 值乘以： $\sqrt{\frac{1}{n[l-1]}}$ ，相应 python 伪代码：

$$w[l] = \text{np.random.rand}(n[l], n[l-1]) * \text{np.sqrt}(\frac{2}{n[l-1]})$$

3.5 Mini-batch 梯度下降

前面神经网络训练过程是对所有 m 个样本，称为 batch。如果 m 很大，例如达到百万数量级，训练速度往往会很慢，因为每次迭代都要对所有样本进行求和运算和矩阵运算。我们将这种梯度下降算法称为 Batch Gradient Descent。为了解决这一问题，我们可以把 m 个训练样本分成若干个子集，称为 mini-batches，这样每个子集包含的数据量就小了，然后每次在单一子集上进行神经网络训练，速度就会大大提高。这种梯度下降算法叫做 Mini-batch Gradient Descent。

Mini-batches Gradient Descent 的实现过程是先将总的训练样本分成 T 个子集（mini-batches），然后对每个 mini-batch 进行神经网络训练，包括 Forward Propagation, Compute Cost Function, Backward Propagation，循环至 T 个 mini-batch 都训练完毕。

经过 T 次循环之后，所有 m 个训练样本都进行了梯度下降计算。这个过程，我们称之为经历了一个 epoch。对于 Batch Gradient Descent 而言，一个 epoch 只进行一次梯度下降算法；而 Mini-Batches Gradient Descent，一个 epoch 会进行 T 次梯度下降算法。

值得一提的是，对于 Mini-Batches Gradient Descent，可以进行多次 epoch 训练。而且，每次 epoch，最好是将总体训练数据重新打乱、重新分成 T 组 mini-batches，这样有利于训练出最佳的神经网络模型。

一般来说，如果总体样本数量 m 不太大时，例如 $m < 2000$ ，建议直接使用 Batch gradient descent。如果总体样本数量 m 很大时，建议将样本分成许多 mini-batches。推荐常用的 mini-batch size 为 64, 128, 256, 512。这些都是 2 的幂。之所以这样设置的原因是计算机存储数据一般是 2 的幂，这样设置可以提高运算速度。

我们可以这样理解：对于训练样本数特别多的训练样本，可以一次使用一部分，并不需要一次就使用全部的训练数据，这样依然可以有很好的训练效果，关键是让训练变得可行。因为数据量特别大的化，可能训练花费很多资源，是不可行的。

3.6 梯度下降优化

3.6.1 Momentum

动量梯度下降算法，其速度要比传统的梯度下降算法快很多。做法是在每次训练时，对梯度进行指数加权平均处理，然后用得到的梯度值更新权重 W 和常数项 b 。

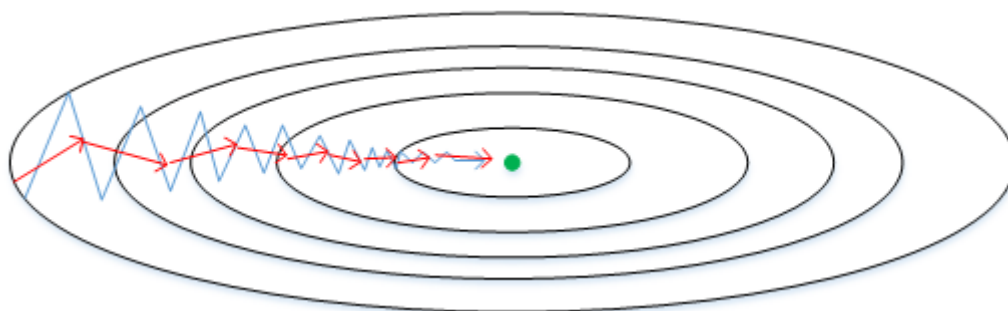


图 3-2 梯度下降过程图

原始的梯度下降算法如上图蓝色折线所示。在梯度下降过程中，梯度下降的振荡较大，尤其对于 W 、 b 之间数值范围差别较大的情况。此时每一点处的梯度只与当前方向有关，产生类似折线的效果，前进缓慢。而如果对梯度进行指数加权平均，这样使当前梯度不仅与当前方向有关，还与之前的方向有关，这样处理让梯度前进方向更加平滑，减少振荡，能够更快地到达最小值处。计算过程如下：

计算 dW, db

$$V_{dW} = \beta V_{dW} + (1 - \beta)dW$$

$$V_{db} = \beta V_{db} + (1 - \beta)db$$

$$W = W - \alpha V_{dW}, b = b - \alpha V_{db}$$

一般设置 $\beta = 0.9$ ，即当前层占 0.1 的比例，其他层占 0.9 的比例。

3.6.2 RmSprop

RMSprop 是另外一种优化梯度下降速度的算法。每次迭代训练过程中， W 和常数项 b 的更新表达式为：

$$S_w = \beta S_{dW} + (1 - \beta)dW^2$$

$$S_b = \beta S_{db} + (1 - \beta)db^2$$

$$W = W - \alpha \frac{dW}{\sqrt{S_w} + \epsilon}, b = b - \alpha \frac{db}{\sqrt{S_b} + \epsilon}$$

为了避免 RMSprop 算法中分母为零，通常可以在分母增加一个极小的常数 $\epsilon = 10^{-8}$ 。

在图 3-2 中，为了便于分析，令水平方向为 W 的方向，垂直方向为 b 的方向。可以看到梯度下降在垂直方向上振荡较大，在水平方向上振荡较小，表示在 b 方向上梯度较大，即 db 较大，而在 W 方向上梯度较小，即 dW 较小。因此 S_w 较小， S_b 较大。在更新 W 和 b 的表达式中，变化值 $\frac{dW}{\sqrt{S_w} + \epsilon}$ 较大， $\frac{db}{\sqrt{S_b} + \epsilon}$ 较小。也就使得 W 变化得多一些， b 变化得少一些。即加快了 W 方向的速度，减小了 b 方向的速度，减小振荡，实现快速梯度下降算法，

3.6.3 Adam

Adam (Adaptive Moment Estimation) 算法结合了动量下降算法和 RMSprop 算法。每 t 次迭代训练过程中， W 和常数项 b 的更新表达式为：

$$V_{dW} = 0, S_{dW} = 0, V_{db} = 0, S_{db} = 0$$

$$V_{dW} = \beta_1 V_{dW} + (1 - \beta_1)dW, V_{db} = \beta_1 V_{db} + (1 - \beta_1)db$$

$$\begin{aligned}
V_{dW}^{corrected} &= \frac{V_{dW}}{1 - \beta_1^t}, V_{db}^{corrected} = \frac{V_{db}}{1 - \beta_1^t} \\
S_{dW}^{corrected} &= \frac{S_{dW}}{1 - \beta_2^t}, S_{db}^{corrected} = \frac{S_{db}}{1 - \beta_2^t} \\
W &= W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected} + \varepsilon}}, b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected} + \varepsilon}}
\end{aligned}$$

Adam 算法包含了几个超参数,分别是 α , β_1 , β_2 , ε 。其中 β_1 通常设置为 0.9, β_2 通常设置为 0.999, ε 通常设置为 10^{-8} 。实际应用中, Adam 算法是最常用的梯度下降优化算法,它结合了动量梯度下降和 RMSprop 各自的优点,使得神经网络训练速度大大提高。

3.7 Batch Normalization

Sergey Ioffe 和 Christian Szegedy 两位学者提出了 Batch Normalization 方法。Batch Normalization 不仅可以让调试超参数更加简单,而且可以让神经网络模型更加“健壮”。也就是说较好模型可接受的超参数范围更大一些,包容性更强,使得更容易去训练一个深度神经网络。

在神经网络中,第 1 层隐藏层的输入是第 l-1 层隐藏层的输出 $A^{[l-1]}$,对 $A^{[l-1]}$ 进行标准化处理,可以提高 W 和 b 的训练速度和准确度。实际应用中,一般是对 $Z^{[l-1]}$ 进行标准化处理而不是 $A^{[l-1]}$ 。对于第 l 层隐藏层的输入 $Z^{[l-1]}$ 做如下标准化处理, i 表示样本,忽略上标[l-1]:

$$\begin{aligned}
\mu &= \frac{1}{m} \sum_{i=1}^m Z^{(i)} \\
\sigma^2 &= \frac{1}{m} \sum_{i=1}^m (Z_i - \mu)^2 \\
Z_{norm}^{(i)} &= \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \\
\tilde{Z}^{(i)} &= \gamma Z_{norm}^{(i)} + \beta
\end{aligned}$$

$Z_{norm}^{(i)}$ 的均值为 0, 方差为 1。我们引入 γ 、 β , 作用是让 $\tilde{Z}^{(i)}$ 的均值和方差为

任意值，只需调整其值就可以了，类似于 W 和 b 一样，可以通过梯度下降等算法求得。

Batch Norm 不仅能够提高神经网络训练速度，而且能让神经网络的权重 W 的更新更加“稳健”，尤其在深层神经网络中更加明显。比如神经网络很后面的 W 对前面的 W 包容性更强，即前面的 W 的变化对后面 W 造成的影响很小，整体网络更加健壮。Batch Norm 减少了各层 $W^{[l]}$, $B^{[l]}$ 之间的耦合性，让各层更加独立，实现自我训练学习的效果。

第4章 卷积神经网络

传统深度神经网络一个问题是参数过多，而卷积神经网络解决了参数过多的问题。卷积神经网络主要应用于图像识别，也有人提出卷积神经网络和循环神经网络相结合的 NLP 算法，这一章就简单介绍卷积神经网络中三中网络层：卷积层、Pooling 层、全连接层的原理的公式。

4.1 卷积层

两个矩阵做卷积操作计算过程如下，下图 4-1 显示了卷积后的第一个值和最后一个值：

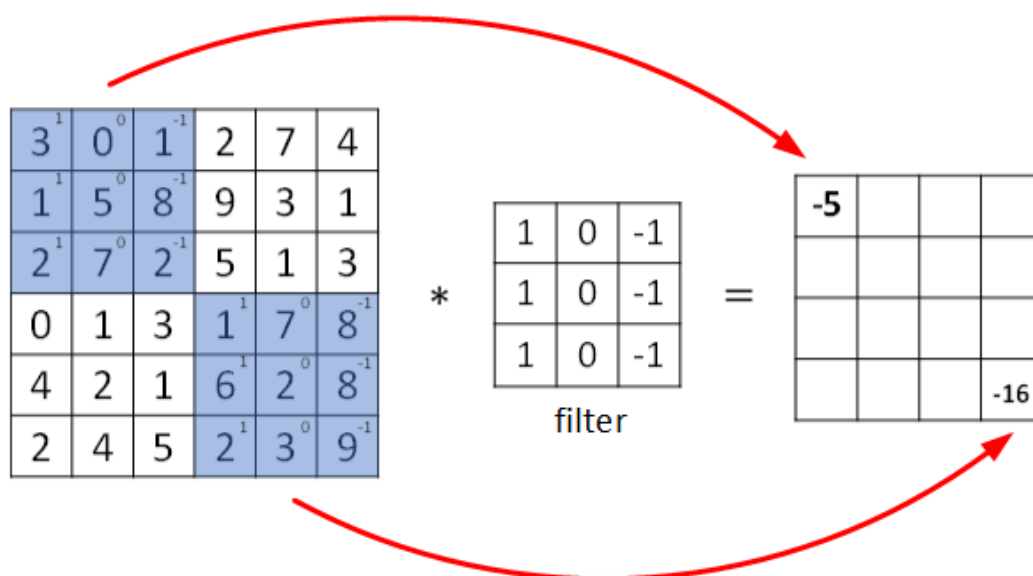


图 4- 1 卷积

4.1.1 Padding

从 4-1 可以看到，原始矩阵尺寸是(6,6)，经过卷积变为(5,5)，这样会有两个问题：1.输出尺寸变小，2.输入的边缘信息丢失。可以使用 padding 方法，把原始矩阵进行扩展，扩展区域补零，用 p 来表示每个方向扩展的宽度。

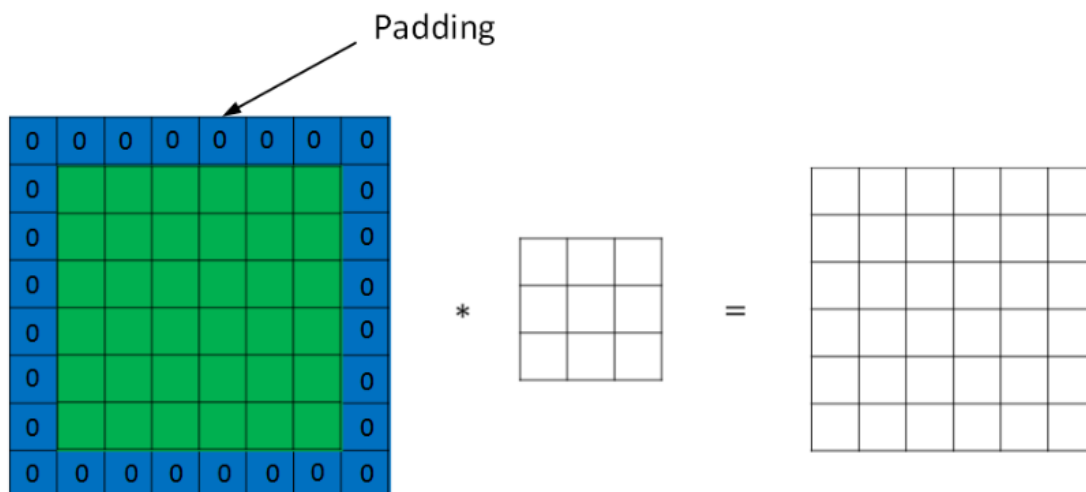


图 4- 2 padding

经过 padding 后，原始矩阵尺寸为 $(n+f-1, n+f-1)$ ，卷积核的尺寸为 (f, f) ，经过

卷积，输出和输入的矩阵尺寸一样，我们称为“same convolutions”。

4.1.2 步长

步长是卷积核在输入矩阵中水平方向和垂直方向每次的步进长度。之前我们默认 stride=1。若 stride=2，则表示 filter 每次步进长度为 2，即隔一点移动一次。

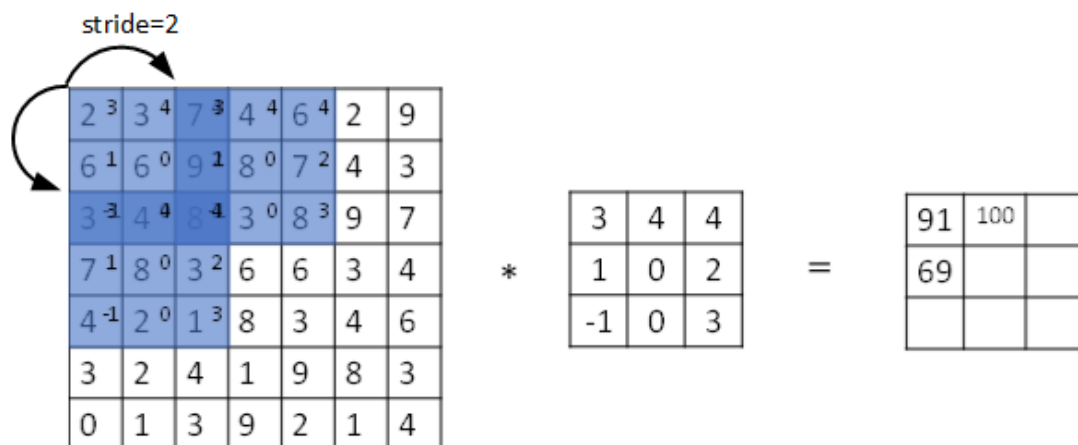


图 4-2 步长为 2 的卷积

4.1.3 多通道卷积

卷积神经网络的输入可以是多通道的。多通道矩阵的卷积运算和单通道矩阵卷积运算一致。过程是将每个单通道与对应的 filter 进行卷积运算求和，然后再将 3 通道的和相加，得到输出图片的一个像素值。

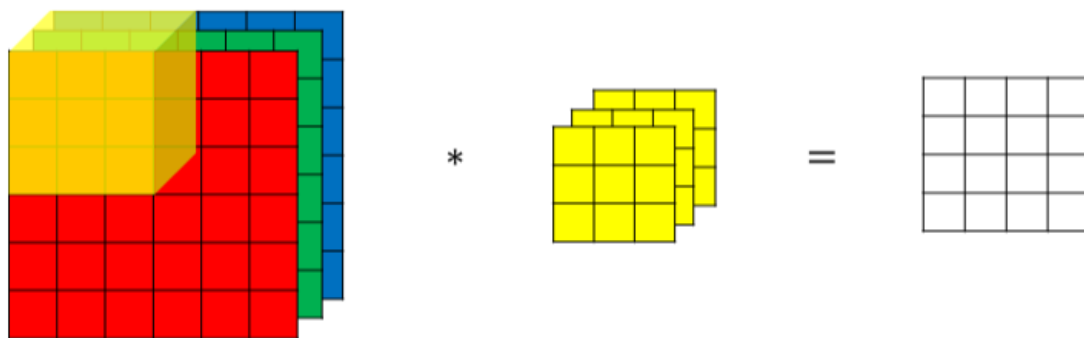


图 4-3 多通道卷积

4-3 中一个卷积核的通道数和输入矩阵的通道数相同。可以有多个卷积核，下图 4-4 中有两个卷积核，每个卷积核的通道数和输入矩阵的卷积数相同。

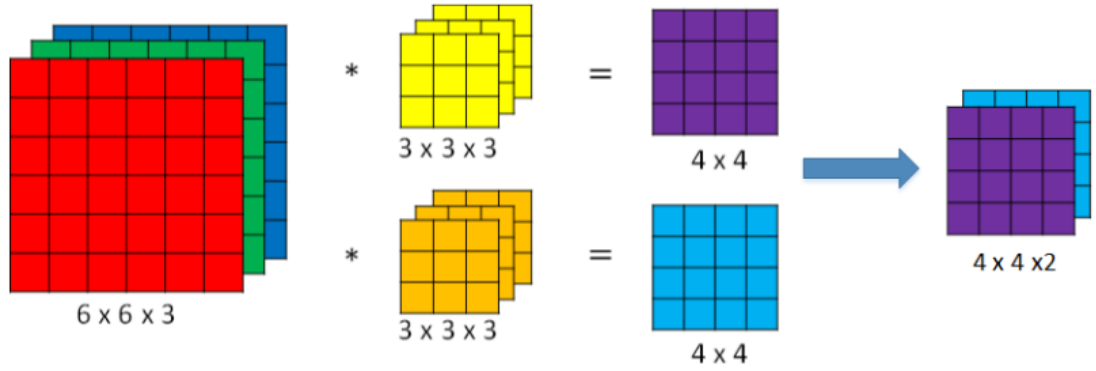


图 4-4 多通道多卷积核

4.1.4 一个卷积层

设层数为 l ，卷积核的长和宽是相等的，卷积核的边长： $f^{[l]}$ ，padding： $p^{[l]}$ ，步长： $s^{[l]}$ ，卷积核数量： $n_c^{[l]}$ 。

对于卷积层 l ：

输入维度为： $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

每个 filter 的维度为： $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

权重维度为： $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

输出维度为： $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

其中， $n_H^{[l]} = \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$ ， $n_W^{[l]} = \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$

4.2 Pooling 层

Pooling layers 是 CNN 中用来减少宽度和长度，保持通道数不变，提高运算速度的，同样能减小 noise 影响，让各特征更具有健壮性。它是在滤波器算子滑动区域内取最大值，即 max pooling，这是最常用的做法。

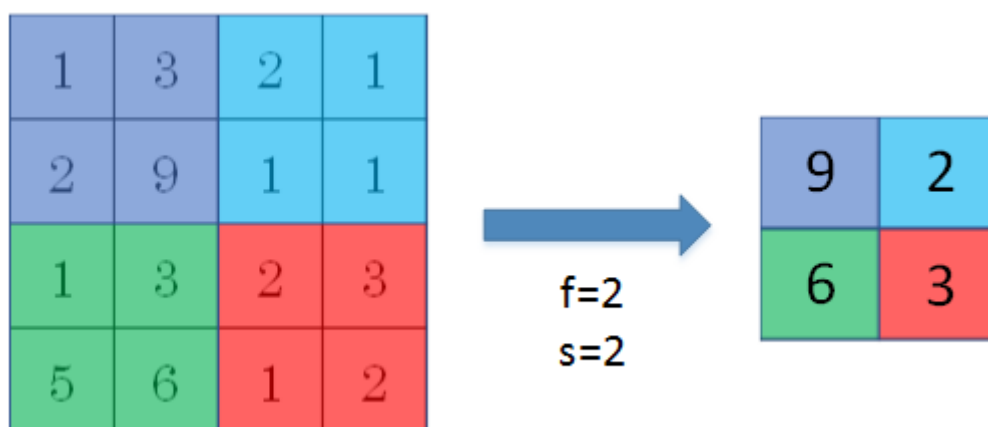


图 4- 4 Max Pooling

Max pooling 的好处是只保留区域内的最大值(特征),忽略其它值,降低 noise 影响,提高模型健壮性。而且, max pooling 需要的超参数仅为滤波器尺寸 f 和滤波器步进长度 s , 没有其他参数需要模型训练得到, 计算量很小。

如果是多个通道, 那么就每个通道单独进行 max pooling 操作。除了 max pooling 之外, 还有一种做法: average pooling。顾名思义, average pooling 就是在滤波器算子滑动区域计算平均值。实际应用中, max pooling 比 average pooling 更为常用。

4.3 全连接层

全连接层是深度神经网络, 当模型输入的矩阵, 长度、宽度、通道减小到一定数目时, 把他们排列成 1 列, 维度为 $(n_x, 1)$ 。后面通过数个神经网络层来减少输入尺寸的大小, 最后加上输出层 (多分类使用 softmax, 二分类使用 sigmoid)。

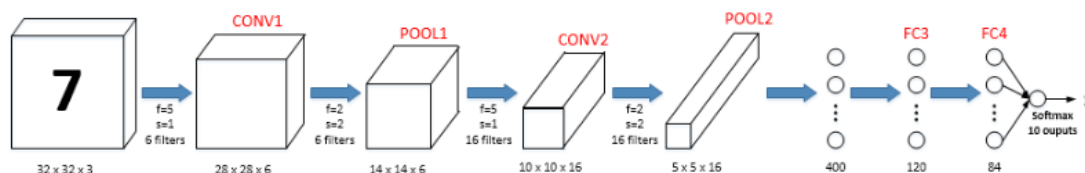


图 4- 5 CNN 多分类

图 4-5 中, CON 层后面紧接一个 POOL 层, CONV1 和 POOL1 构成第一层, CONV2 和 POOL2 构成第二层。特别注意的是 FC3 和 FC4 为全连接层 FC, 它

跟标准的神经网络结构一致。最后的输出层（softmax）由 10 个神经元构成。

整个网络各层的尺寸和参数如下图 4-6 所示：

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3072	0
CONV1(f=5,s=1)	(28,28,6)	4704	158
POOL1	(14,14,6)	1176	0
CONV2(f=5,s=1)	(10,10,16)	1600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48120
FC4	(84,1)	84	10164
Softmax	(10,1)	10	850

图 4- 6 模型参数

第5章 循环神经网络

循环神经网络是针对序列模型的一种有效深度学习算法。本章介绍了 RNN 的正向传播和反响传播，以及数种经过优化的 RNN：GRU、LSTM、BRNN、DRNNs。

5.1 循环神经网络模型

下面使用命名实体识别为例，例如找出这个句子 “Harry Potter and Hermione Granger invented a new spell.” 中的人名。句子中有 9 个单词，输出 y 即为 1×9 向量，对于输入 x ，可以表示为：

$$[x^{(1)} x^{(2)} x^{(3)} x^{(3)} x^{(4)} x^{(5)} x^{(6)} x^{(7)} x^{(8)} x^{(9)}]$$

输出 $y = [1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0]$ 。如果使用标准的深度神经网络，其模型结构如下：

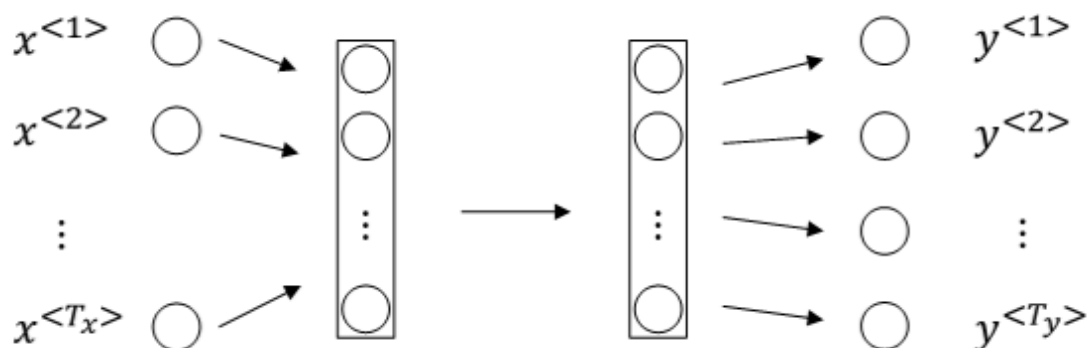


图 5- 1 DNN 处理序列信息

这样有两个问题：第一，不同的样本的输入序列长度或输出序列长度不同，造成模型难以统一。解决办法之一是设定一个最大序列长度，对每个输入和输出序列补零并统一到最大长度。但是这种做法实际效果并不理想。第二个问题，也是主要问题，这种标准神经网络结构无法共享序列不同输入之间的特征。例如，如果某个 $x^{<1>}$ 即“Harry”是人名成分，那么句子其它位置出现了“Harry”，也很可能也是人名。标准的神经网络不适合解决序列模型问题，而循环神经网络(RNN)是专门用来解决序列模型问题的。RNN 模型结构如下：

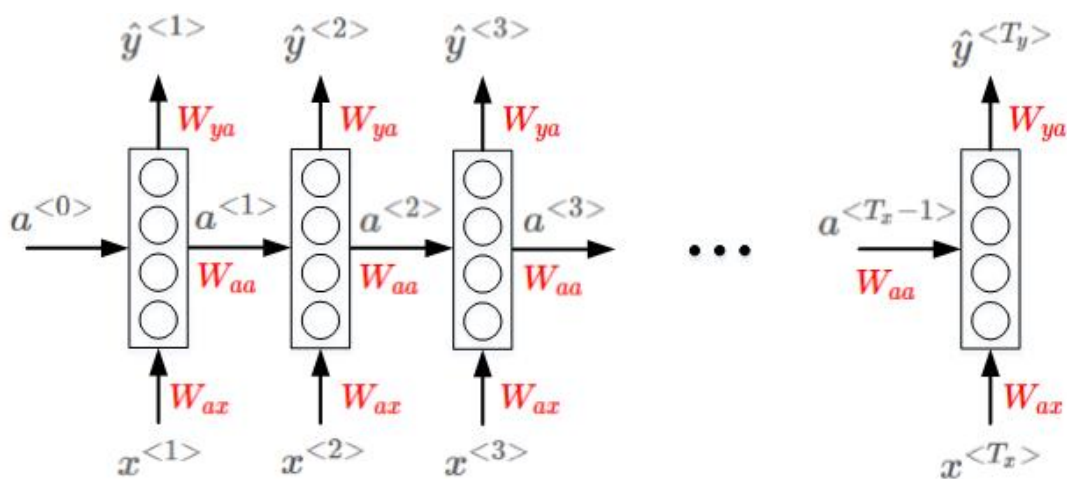


图 5- 2 RNN 模型

5.1.1 正向传播

RNN 正向传播过程为：

$$a^{(t)} = g(W_{aa}a^{(t-1)} + W_{ax}x^{(t)} + ba)$$

$$\hat{y}^{(t)} = g(W_{ya}a^{(t)} + b_y)$$

为了简化表达式，可以进行整合：

$$W_{aa}a^{(t-1)} + W_{ax}x^{(t)} + ba = [W_{aa} \ W_{ax}] \begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix} = W_a[a^{(t-1)}, x^{(t)}]$$

则正向传播可以表示为：

$$a^{(t)} = g(W_a[a^{(t-1)}, x^{(t)}] + ba)$$

$$\hat{y}^{(t)} = g(W_{ya}a^{(t)} + b_y)$$

5.1.2 反向传播

反向传播需要先计算出损失函数的导数，再由右向左分别计算损失函数对参数 W_a, W_y, b_a, b_y 的偏导数。思路与做法与标准的神经网络是一样的。一般可以通过成熟的深度学习框架自动求导，例如 PyTorch、Tensorflow 等。这种从右到左的求导过程被称为 Backpropagation through time。

5.2 RNN 的梯度消失

语句中可能存在跨度很大的依赖关系，即某个 word 可能与它距离较远的某个 word 具有强依赖关系。例如下面这两条语句：

The *cat*, which already ate fish, *was* full.

The *cats*, which already ate fish, *were* full.

第一句话中，*was* 受 *cat* 影响；第二句话中，*were* 受 *cats* 影响。它们之间都跨越了很多单词。而一般的 RNN 模型每个元素受其周围附近的影响较大，难以建立跨度较大的依赖性。上面两句话的这种依赖关系，由于跨度很大，普通的 RNN 网络容易出现梯度消失，捕捉不到它们之间的依赖，造成语法错误。

5.3 Gated Recurrent Unit (GRU)

GRU 改善了梯度下降的问题，它比一般的循环神经网络要复杂。一个使用 \tanh 为激活函数的 RNN 的隐藏层单元结构如下图 5-3 所示：

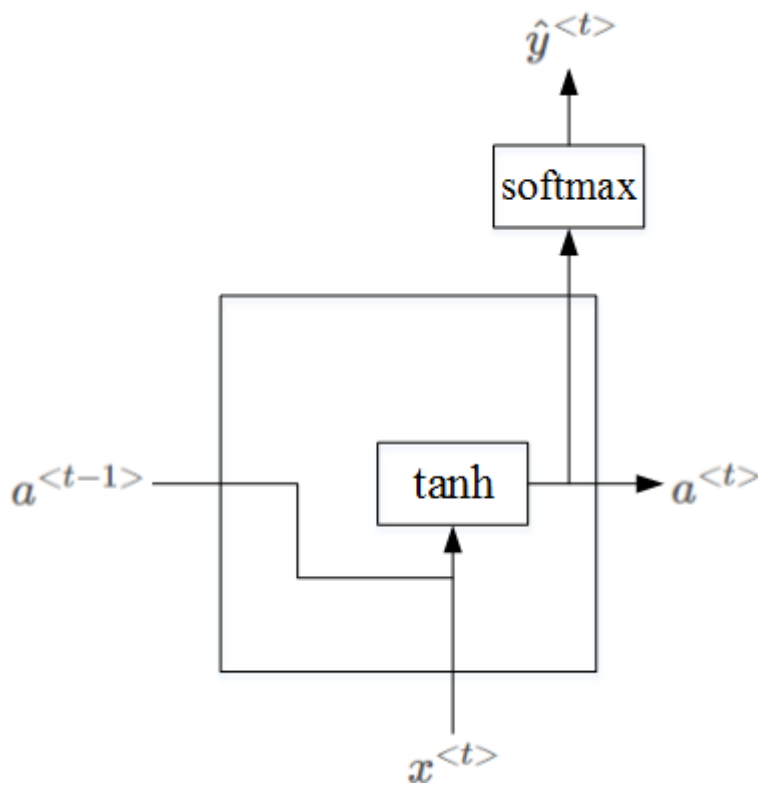


图 5- 4 RNN 单元模型

$$a^{(t)} = \tanh(W_a[a^{(t-1)}, x^{(t)}] + b_a)$$

为了解决梯度消失问题，对上述单元进行修改，添加了记忆单元，构建 GRU，如下图所示：

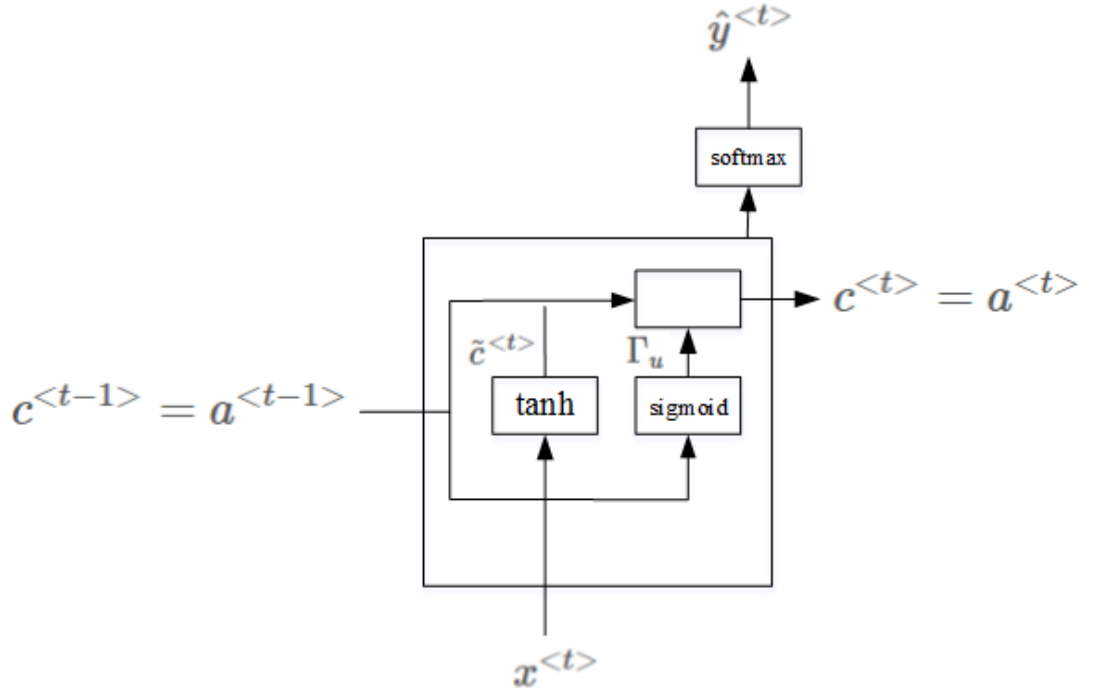


图 5- 5 GUR 单元模型

$$\check{c}^{(t)} = \tanh(W_a[a^{(t-1)}, x^{(t)}] + b_a)$$

$$\Gamma_u = \sigma(W_u[c^{(t-1)}, x^{(t)}] + b_u)$$

$$c^{(t)} = \Gamma_u * \check{c}^{(t)} + (1 - \Gamma_u) * c^{(t-1)}$$

Γ_u 意为 **gate**，记忆单元，当 $\Gamma_u = 1$ 时，代表更新，当 $\Gamma_u = 0$ 时，代表记忆，保留之前的模块输出。因此， Γ_u 能够保证 RNN 模型中跨度很大的依赖关系不受影响，消除梯度消失问题。实际使用中，我们在增加一个参数 Γ_r ：

$$\Gamma_r = \text{sigmoid}(W_r[c^{(t-1)}, x^{(t)}] + b_r)$$

$$\check{c}^{(t)} = \tanh(\Gamma_r * W_a[a^{(t-1)}, x^{(t)}] + b_a)$$

$$\Gamma_u = \sigma(W_u[c^{(t-1)}, x^{(t)}] + b_u)$$

$$c^{(t)} = \Gamma_u * \check{c}^{(t)} + (1 - \Gamma_u) * c^{(t-1)}$$

$$a^{(t)} = c^{(t)}$$

注意以上表达式中的 * 表示元素相乘，而非矩阵相乘。

5.4 Long Short Term Memory (LSTM)

LSTM 是另一种更强大的解决梯度消失问题的方法。它对应的 RNN 隐藏层单元结构如下图所示：

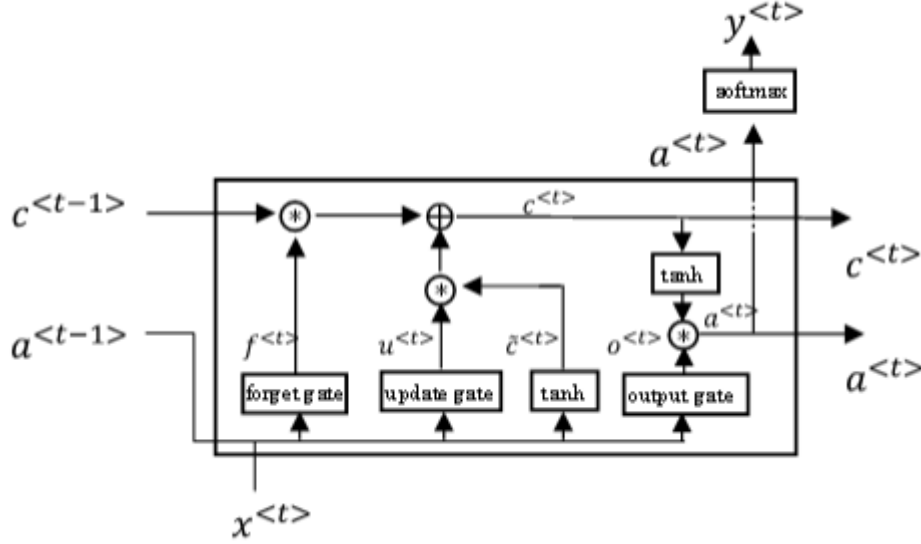


图 5- 6 LSTM 单元模型

LSTM 包含三个 gates: Γ_u , Γ_f , Γ_o , 分别对应 update gate, forget gate, output gate。其中 update gate 表示是否使用备用输入, forget gate 表示是否使用原输入, output gate 表示如何将 $c^{(t)}$ 变为 $a^{(t)}$ 。

$$\check{c}^{(t)} = \tanh(W_c[a^{(t-1)}, x^{(t)}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{(t-1)}, x^{(t)}, c^{(t-1)}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{(t-1)}, x^{(t)}, c^{(t-1)}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{(t-1)}, x^{(t)}, c^{(t-1)}] + b_o)$$

$$c^{(t)} = \Gamma_u * \check{c}^{(t)} + \Gamma_f * c^{(t-1)}$$

$$a^{(t)} = \Gamma_o * c^{(t)}$$

GRU 可以看成简化的 LSTM, 两种方法都具有各自的优势。

5.5 Bidirectional RNN

BRNN 能够同时对序列进行双向处理，性能大大提高。它的结构如下图 5-7

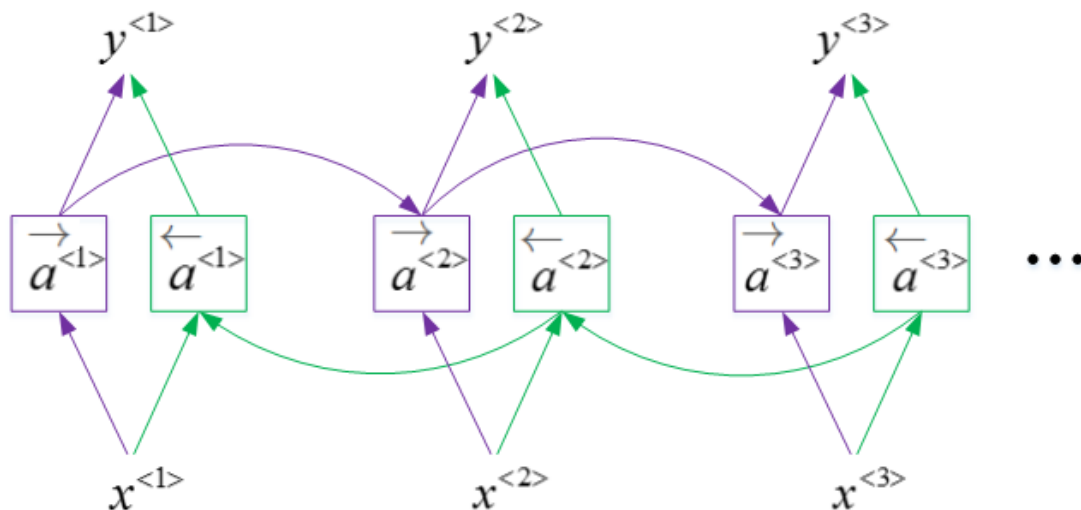


图 5-7 双向 RNN

BRNN 对应的输出 $y^{(t)}$

$$\hat{y}^{(t)} = g(W_y[\vec{a}^{(t)}, \tilde{a}^{(t)}] + b_y)$$

5.6 Deep RNNs

Deep RNNs 由多层 RNN 组成，其结构如下图所示：

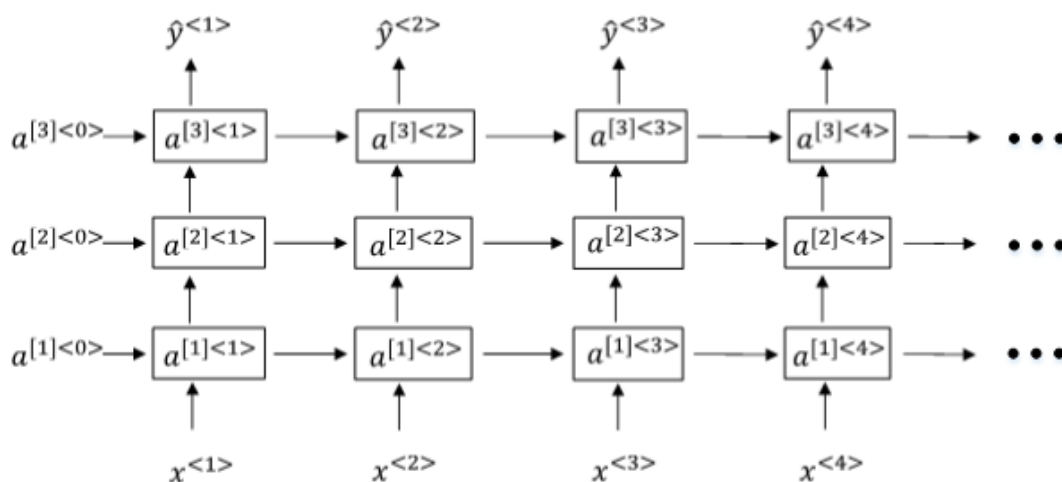


图 5-8 深度 RNN

和 DNN 一样，层数用上标 $[l]$ 表示，Deep RNNs 中 $a^{[l](t)}$ 的表达式：

$$a^{[l](t)} = g\left(W_a^{[l]}[a^{[l](t-1)}, a^{[l-1](t)}] + b_a^{[l]}\right)$$

DNN 层数可达 100 多，而 Deep RNNs 一般没有那么多层，3 层 RNNs 已经较复杂了。

第6章 自然语言处理

本章主要讲解了自然语言处理的历史和常用模型，引出了作为训练 SQuAD 数据集的模型 BERT。

6.1 NLP 发展历史

早期的自然语言处理和深度神经网络没有关系，2003 年第一篇将深度神经网络和 NLP 相结合的论文出现了，而这篇论文当时并没有引起太大反响。在 2010 年前，使用深度神经网络并不是 NLP 的主流。自从 2012 年来，图像识别领域最重要的比赛 ImageNet 的冠军和排名靠前的模型，都变为卷积神经网络模型。而这一现实也让 NLP 研究者更加关注深度学习，2013 年 Word2Vec 被提出并开源，随后有了各种针对训练词向量的优化方法被提出。2017 年 Google 发表 Attention is All You Need 论文，介绍了另一种算法 transformer，随后使用 transformer 的 GPT 和 Bert 在各种 NLP 比赛的表现比使用 LSTM 模型更好，这几乎标志这以后的 NLP 不会在使用 RNN，会使用 transformer 这种类似于半监督模型。

自然语言处理有四大类任务：

1. 序列标注，这是最典型的 NLP 任务，比如中文分词，词性标注，命名实体识别，语义角色标注等都可以归入这一类问题，它的特点是句子中每个单词要求模型根据上下文都要给出一个分类类别。
2. 分类任务，比如我们常见的文本分类，情感计算等都可以归入这一类。它的特点是不管文章有多长，总体给出一个分类类别即可
3. 句子关系判断，比如 Entailment, QA, 语义改写，自然语言推理等任务都是

这个模式，它的特点是给定两个句子，模型判断出两个句子是否具备某种语义关系。

4. 生成式任务，比如机器翻译，文本摘要，写诗造句，看图说话等都属于这一类。它的特点是输入文本内容后，需要自主生成另外一段文字。

6.2 Word representation

自然语言有一些特点，比如词语是有限的，词语与词语之间有一些关系。电脑并不知道人类词语的含义，需要使用二进制数字表示词语。最开始使用 one-hot 的方式对每个单词进行编码。如果一个单词库有 10000 个单词，那么每个单词的 one-hot 向量是(10000,1)的，10000 个数中只有 1 个是 1，9999 个数是 0。

这中 one-hot 表征单词的方法最大的缺点就是每个单词都是独立的、正交的，无法知道不同单词之间的相似程度。例如 Apple 和 Orange 都是水果，词性相近，但是单从 one-hot 编码上来看，内积为零，无法知道二者的相似性。在 NLP 中，我们更希望能掌握不同单词之间的相似程度。

6.3 Word embeddings

针对上面 one-hot 编码的缺点，可以使用特征表征的方法对每个单词进行编码。2003 年 Bengio 首次提出 word embedding 的概念,中文词嵌入，或者词向量。使用一个特征向量表征单词，特征向量的每个元素都是对该单词某一特征的量化描述，量化范围可以是[-1,1]之间。特征表征的例子如下图所示：

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮	⋮	⋮	⋮	⋮

图 6-1 Word embedding

特征向量的长度依情况而定，特征元素越多则对单词表表征得越全面。这里的特征向量长度设定为 300。使用特征表征之后，词汇表中的每个单词都可以使用对应的 300×1 的向量来表示，该向量的每个元素表示该单词对应的某个特征值。

这种特征表征的优点是根据特征向量能清晰知道不同单词之间的相似程度，例如 Apple 和 Orange 之间的相似度较高，很可能属于同一类别。这种单词“类别”化的方式，大大提高了有限词汇量的泛化能力。这种特征化单词的操作被称为 Word Embeddings，即单词嵌入。

Word embedding 是自然语言处理中的一个迁移学习应用，它的特性使得很多 NLP 任务能方便地进行迁移学习。

6.4 Neural Network Language Model

上面提到了 Word embedding，那么它是怎么训练得到的？2003 年 Bengio 等人发表了一篇开创性的文章：A neural probabilistic language model[3]。也就是 2003 年，才有人使用神经网络进行自然语言处理。NNLM 整个模型的网络结构如下图：

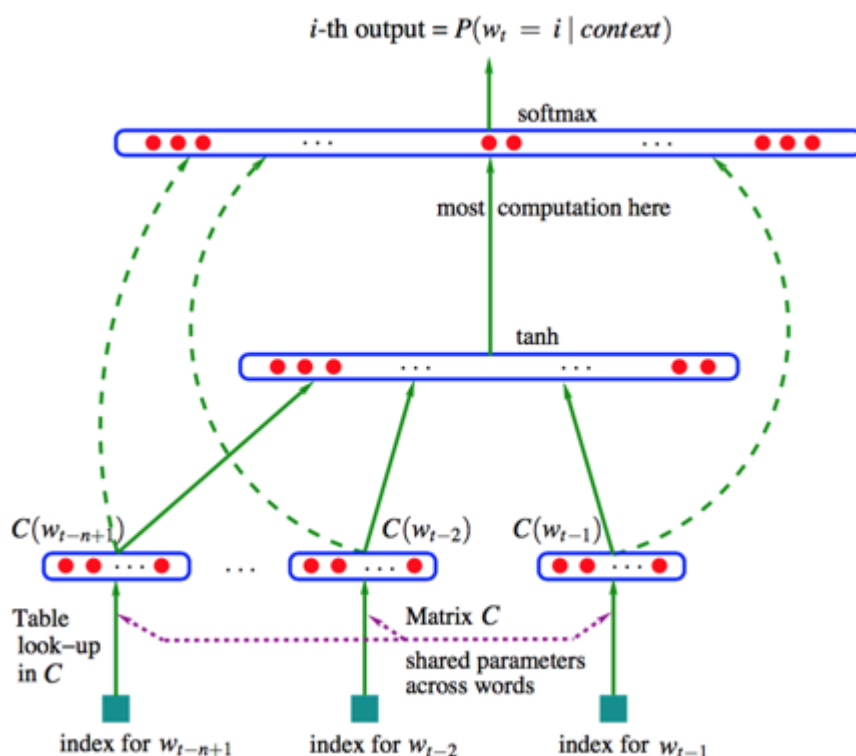


图 6-2 NNLM

这是一个很简单的神经网络。训练方法是在一个句子中，通过第 t 个单词前面 $n-1$ 个单词来预测单词。通过以下几点来理解：

- 输入是第 t 个单词的前面 $n-1$ 个单词的 one-hot 编码，可以认为 one-hot 编码矩阵维度为 $(10000, 1)$ ，图片矩阵 C 就是 word embedding，它的维度是 $(300, 10000)$ ，可以认为矩阵 C 是一个转移矩阵，这样得到 $n-1$ 个维度为 $(300, 1)$ 的矩阵。我们将这 $n-1$ 个矩阵拼接在一起，输入到一个隐藏层。
- 上面就是一个简单的前向反馈神经网络，它由一个 tanh 隐层和一个 softmax 输出层组成。通过将 Embedding 层输出的维度为 $(300 \times (n-1), 1)$ 词向量输入到一个神经元个数为 H 个的隐藏层。输出层使用 softmax，分类的类别就是 10000 个单词
- 每次训练输出 10000 个单词的向量，那么需要最大化 $p(w_t | w_{t-1}, w_{t-2} \dots w_{t-n+1})$

这就是第一个神经网络和自然语言处理相结合的模型，它的缺点是：1. 只能处理定长的序列。2. 训练太慢了，在 tanh 层和 softmax 层计算量很大。

6.5 Word2Vec

2013 年 Mikolov 提出了新的计算词向量的方法。基于 Mikolov 提出的新方法，2013 年 Google 开发并开源了 Word2Vec。这里介绍两种模型：CBow 和 Skip-Gram。针对模型运算量大的问题，Mikolov 引入了两种优化算法：层次 Softmax（Hierarchical Softmax）和负采样（Negative Sampling）。这里主要介绍 CBow 和 Skip-Gram，并不介绍优化算法。

6.5.1 Continuous Bag-of-Words Model (CBow)

它是一种训练方法，主要思想是从一个句子里面把一个词抠掉，用这个词的上下文去预测被抠掉的这个词，网络结构如下图 6-3

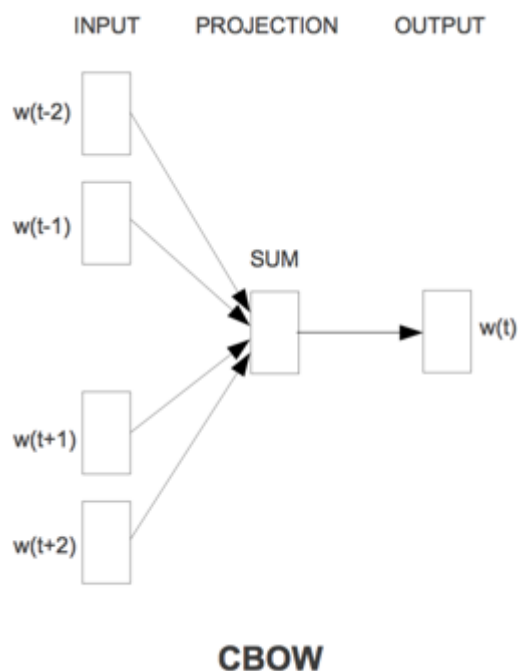


图 6-3 CBOW 示意图

下图 6-4 为 CBOW 的网络结构图

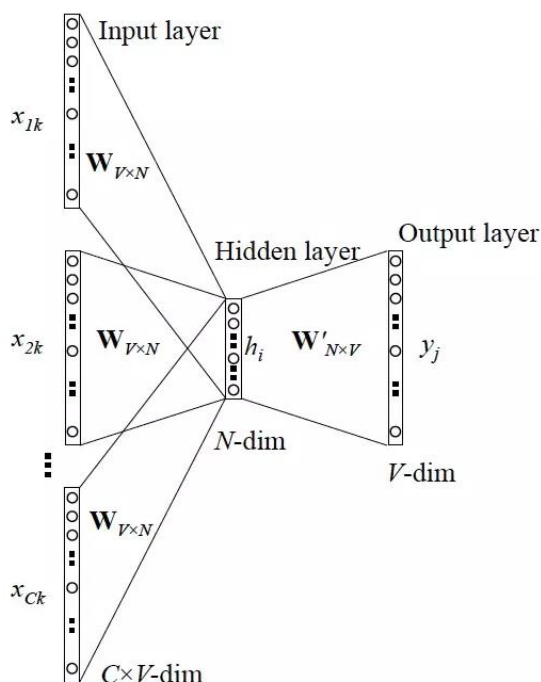


图 6-4 CBOW 网络结构

输入是上下文单词的 one-hot 编码，维度为 $(1, V)$ ， W 是 word embedding，维度为 (V, N) 。输入的 one-hot 编码分别乘以 word embedding 得到对应单词的词向量 $(1, N)$ ，词向量相加求平均作为隐层向量，矩阵维度为 $(1, N)$

$$\text{隐藏层的输入为: } h = \frac{1}{c} W \sum_{i=1}^c x_i$$

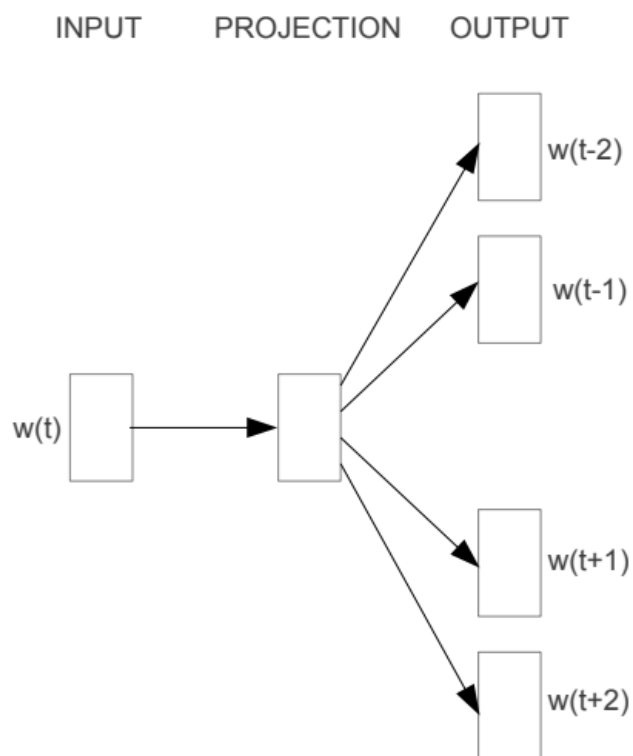
$$\text{隐藏层输出: } u = W'_{N \times V} h$$

使用 softmax 计算概率，然后通过损失函数，梯度下降算法更新 W 和 $W'_{N \times V}$ ， W 就是 Word embedding

可以看出它移除前向反馈神经网络中非线性的 hidden layer，直接将中间层的 Embedding layer 与输出层的 softmax layer 连接；

6.5.2 Skip-Gram

Skip-Gram 模型正好和 CBOW 相反，它用一个词，预测词的上下文。



Skip-gram

图 6-5 Skip-Gram 示意图

它的公式推导这里就不展开了。因为这些获得词向量的方法已经是数年前的，而现在有新的模型。

6.6 ELMo

前面提到的 word embedding 的问题在于无法区分多义词。比如 bank 有两种意思：银行、河岸。ELMo 是一种简单的解决方案。ELMo 的结构是双层双向的 LSTM。

从WE到ELMO：基于上下文的Embedding

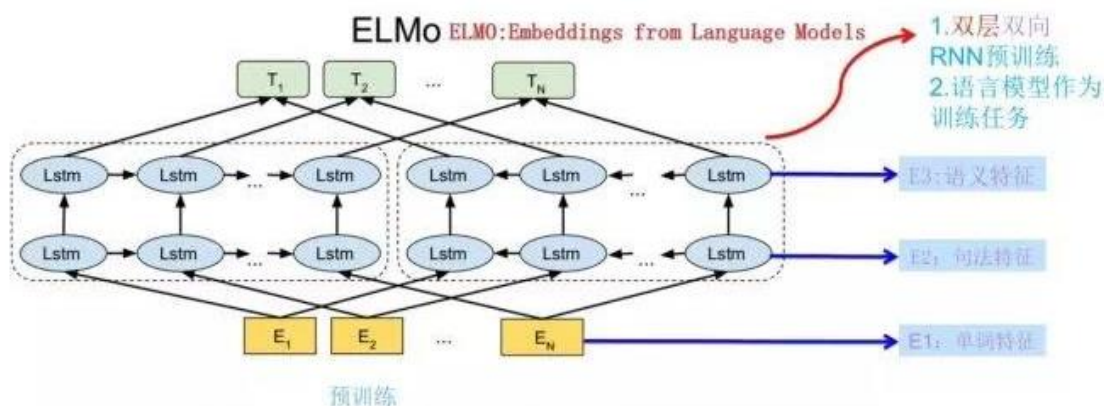


图 6-6 ELMO 示意图

这个模型不仅使用 word embedding 来存储词汇信息，也使用双层双向的 LSTM，也就是在网络中可以存储多义词的语义。

6.7 Transformer

Transformer 出自论文 Attention is All You Need. 下图主体结构图

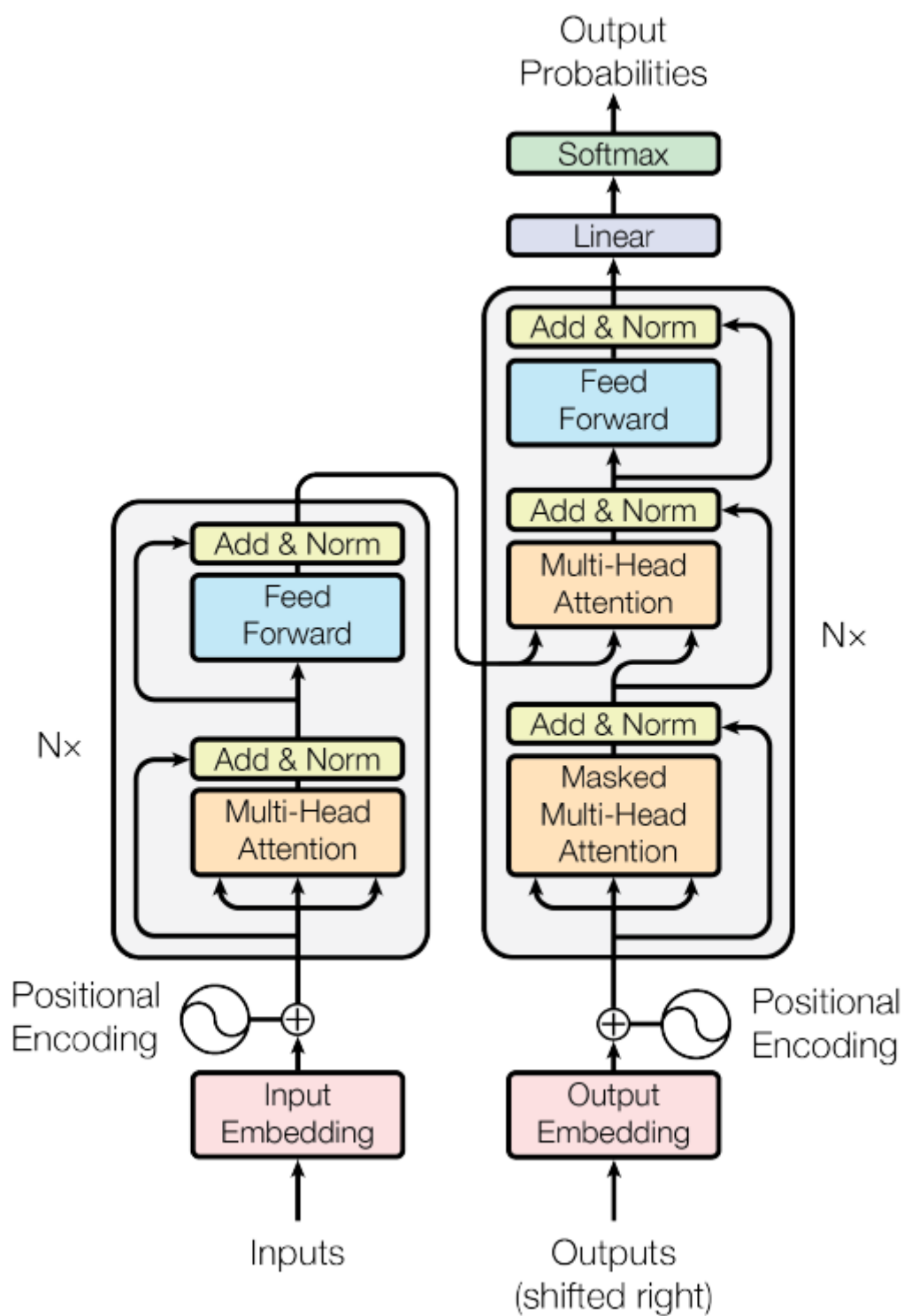


图 6-7 transformer 主体结构图

6.7.1 编码器-解码器

编码器和解码器在机器翻译任务中经常使用。如图 6-5

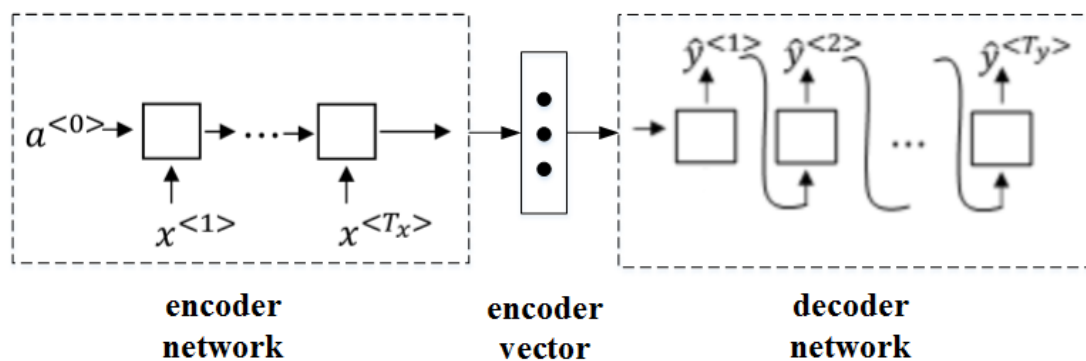


图 6-8 编码器-解码器示意图 1

实际使用中，可以有多个编码器，多个解码器。如下图

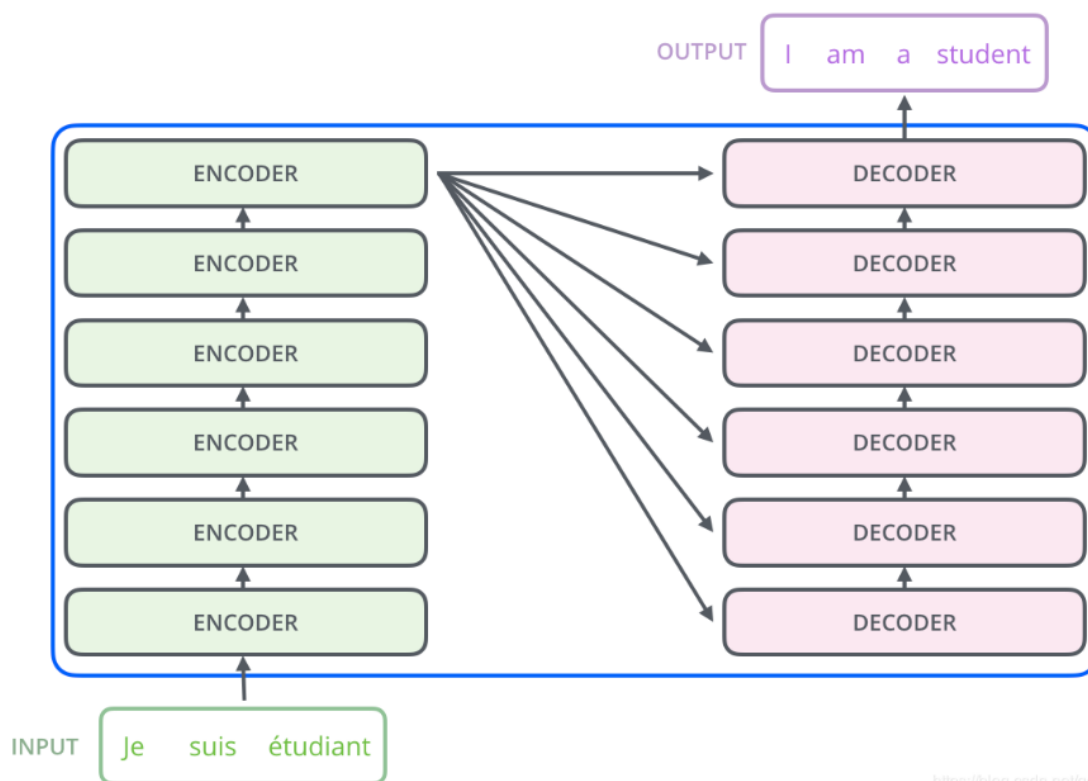


图 6-9 编码器-解码器示意图 2

每一个编码器由两个组件组成：

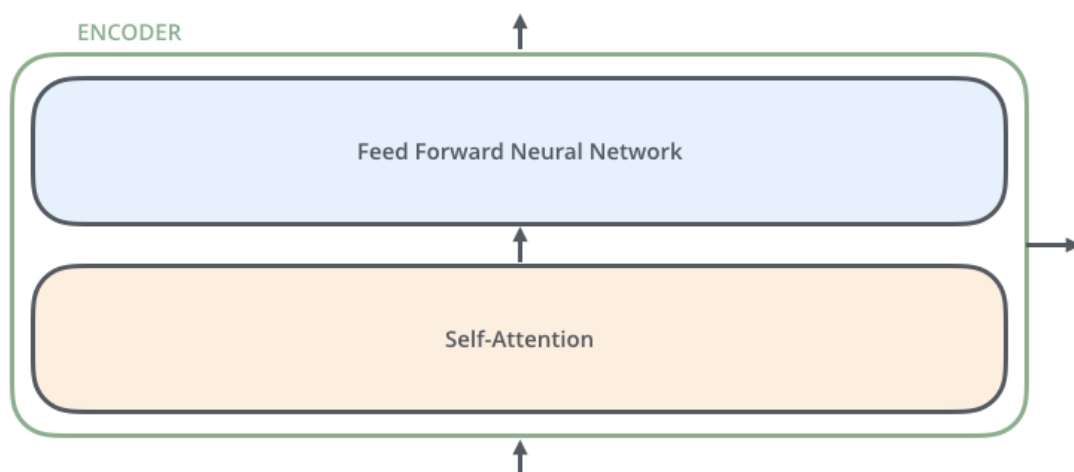


图 6-10 编码器

6.7.2 自注意力

假设我们想要翻译这个句子：“The animal didn't cross the street because it was too tired”。句子中的“it”指“street”还是“animal”？这对人来说很简单，但对计算机很难。自注意力机制可以让“it”和“animal”联系起来。自注意力机制会查看其余词与目标词的关系，。

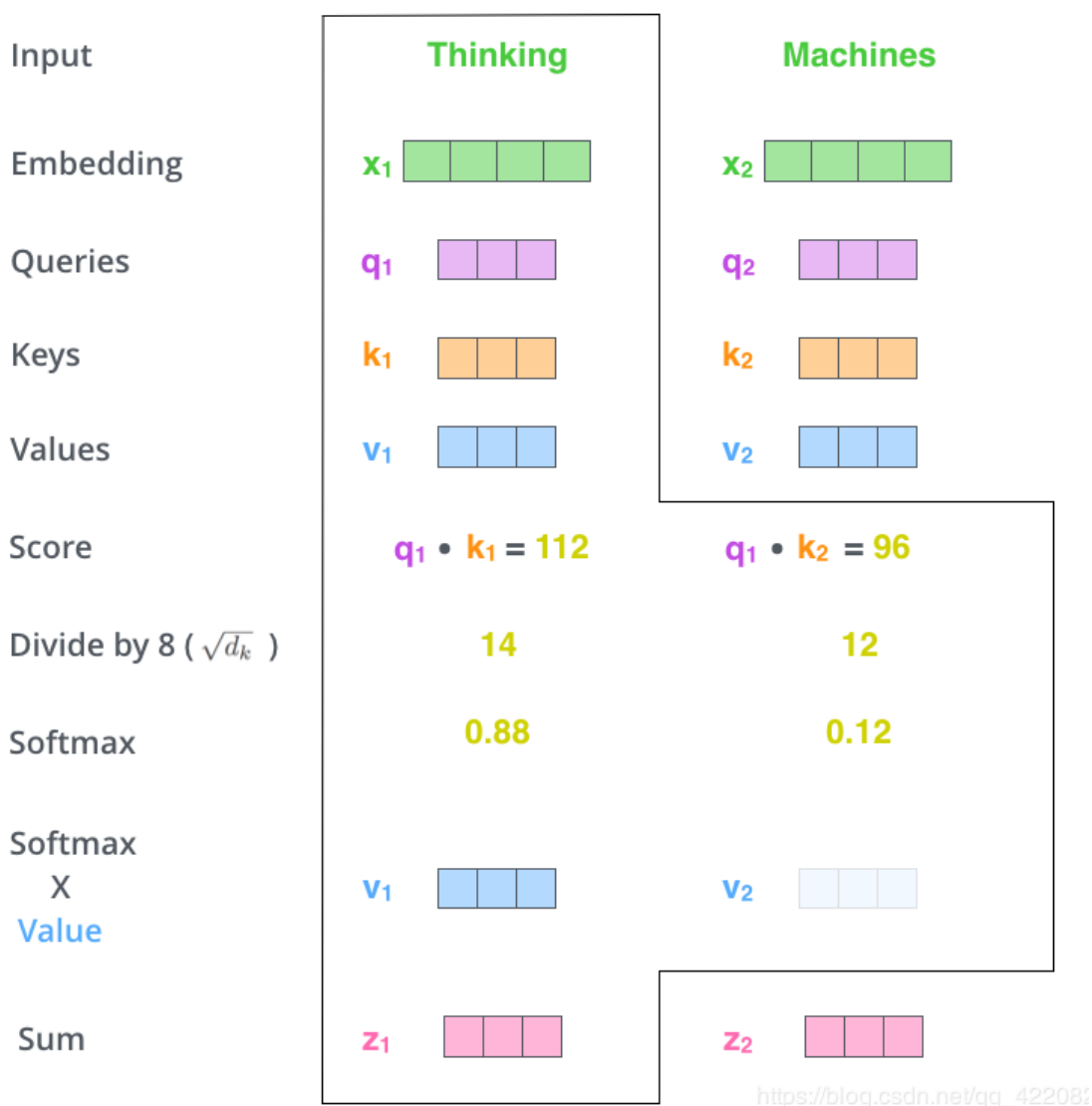


图 6-11 自注意力的计算

- 自注意力机制会对每个输入的词，创建三个矩阵，Query 矩阵，一个 Key 矩阵，一个 Value 矩阵。这三个矩阵是由词嵌入的矩阵乘以我们训练得出的三个矩阵得到的。
- 需要根据这个单词给输入句子中的每一个词打分。当我们编码当前位置的单词的时候，分数决定了我们给其他位置的单词多少关注（权重）。打分通过当前位置单词的 query 矩阵和需要打分位置单词的 key 矩阵点乘得到。
- 将维度除以 8（默认），然后通过 softmax 处理，将值映射到 0-1 之间。
- 每一个 value 矩阵乘以 softmax 处理后的得分，目的是保留重要的词，去

除不相关的词。

- 将权重值向量相加，并输出给前向反馈神经网络。

公式为：m 为句子长度，t 为单词位置。

$$\text{output} = \sum_{i=1}^m \text{softmax}\left(\frac{q_t k_m}{8}\right) v_m$$

6.7.3 多头注意力

多头注意力（multi-headed attention）就是使用多组 Q/K/V 矩阵，各组之间互补相关，这样可以关注到更多的单词。

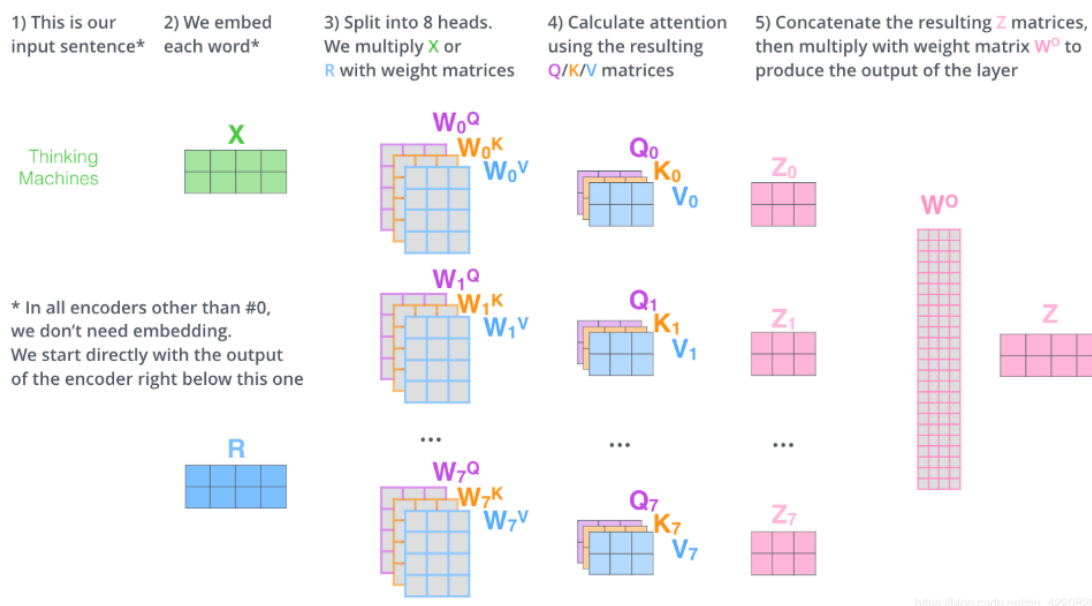


图 6-12 多头自注意力

每个头自注意力都输出一个 Z，那么 n 头自注意力输出 n 个 Z，需要另外乘以另外一个权重矩阵 W^O ，来把这 8 个矩阵组合成 1 个矩阵。

6.8 GPT

GPT 是“Generative Pre-Training”的简称，从名字看它含义是生成式的预训练。它使用了 Transformer，而没有使用 RNN。GPT1.0 在各种数据集上的表现比 ELMo 好，但是没有 bert 表现好，原因是它采用的是单向的语言模型，也就是只

根据一个单词的上本来预测那个单词。一个单词肯定和上下文都有关系，所以单向的语言模型一般没有双向的好，但是由于 Transformer 的效果太好了，GPT1.0 的表现比双向的 ELMo 好。2019 年 2 月 GPT2.0 发布了它使用双向的语言模型，预训练数据集更大，实际效果拭目以待。

6.9 Bert

2018 年 10，Google 开源了 Bert（Bidirectional Encoder Representations from Transformers），它使用了 Transformers，从名字就能看出它是双向的。Bert 是一种预训练模型，它并不是一种新的算法，它使用了 Transformers 算法。接下来的 SQuAD 问答模型就会使用 Bert 来实现。

第7章 SQuAD 数据集的训练

本章给出了 Tensorflow 在 Ubuntu18.04 下的安装过程，SQuAD 数据集格式的解释。下载 Google 在 github 上面的程序来验证回答的正确率。

7.1 环境配置

环境配置主要涉及到 Ubuntu18.04 操作系统、显卡驱动、、CUDA、Tensorflow 的安装。

一、 Ubuntu18.04 操作系统的安装

1. 前期准备

- 1) Ubuntu18.04 镜像文件：ubuntu-18.04.2-desktop-amd64.iso。
- 2) Ultraso 软件：uiso9_cn.exe。
- 3) 压缩卷。
- 4) 制作 U 盘启动盘。

2. 安装 ubuntu18.04 系统

- 1) 修改 BIOS 相关设置。

- 2) 选择启动器：保持 U 盘插入电脑的状态，重启电脑。
- 3) 我的电脑有一个机械硬盘和一个固态硬盘，我将 ubuntu 安装在固态硬盘中，这样电脑中的 Windows10 和 ubuntu 互相不影响。不需要担心引导的问题。

二、 系统更新与显卡驱动安装

1. 系统更新

- 1) 设置→系统设置→软件和更新。
- 2) Ubuntu 软件→Aliyun 源（速度较快）。
- 3) 设置→关于这台计算机→安装更新系统。

2. 显卡驱动的安装

- 1) 设置→系统设置→软件和更新。
- 2) 附加驱动→NVIDIA 专有驱动→应用更改。

三、 Pip3 的安装

1. 安装 pip3: `sudo apt-get install pip3`

Pip3 是针对 python3 的，pip 是针对 python2 的。Ubuntu18.4 默认的 python 版本是 3.6，不同的 Ubuntu 版本支持的 python 版本不同。Python3 已经成为 Ubuntu 重要的一部分，Ubuntu 的图形化界面依赖于 python3，所以不要随便卸载 Python3。另外想要在一个 Ubuntu18.04 上安装 python3.5 是不简单的，因为 Ubuntu 官方并没有给 Ubuntu18.04 提供 python3.5，使用 apt 安装找不到，只能找到 python3.6。

四、 CUDA 的安装

CUDA 是一种通用并行计算架构，能够通过 GPU 快速解决繁琐复杂的计算项目，减少训练和测试时间。

1. 安装 CUDA

- 1) 准备：`cuda_10.0.130_410.48_linux.run`

CUDA Toolkit 10.0 Archive

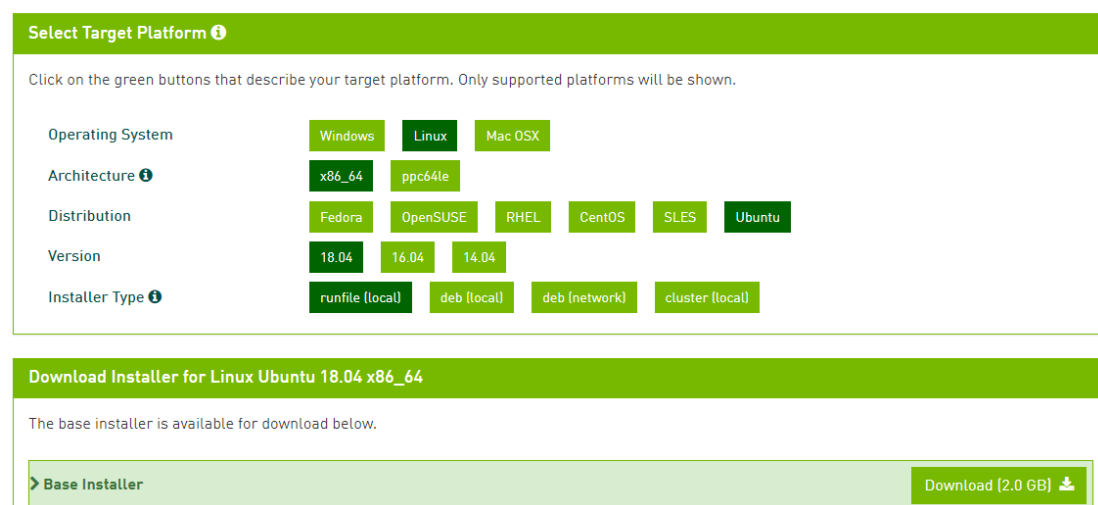


图 7- 1 CUDA 下载页面

使用 Google 或者百度搜索“cuda 下载”，NVIDIA 的官方网站下载 cuda10.0，现在 2019 年 5 月，Tensorflow 最高支持到 cuda10.0，不要下载 cuda10.1

- 2) 安装：`sudo sh cuda_10.0.130_410.48_linux.run`

在安装过程中，会询问是否安装显卡驱动，因为前面已经安装了，这里选择不安装。

- 3) 为了在命令行运行，添加环境变量：

```
sudo gedit ~/.bashrc
```

```
export PATH=/usr/local/cuda-10.0/bin${PATH:+:${PATH}}
```

```
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

2. 测试 CUDA

- 1) 进入目录：`cd ~/NVIDIA_CUDA-10.0_Samples`

- 2) 编译：`sudo make`

- 3) 执行：`sudo ./deviceQuery`

出现自己电脑的 GPU 信息，则 Cuda 安装成功。

3. 添加卷积神经网络加速库 cuDNN

1) 下载

下载三个 deb 包

- cuDNN Runtime Library for Ubuntu18.04 (Deb)
- cuDNN Developer Library for Ubuntu18.04 (Deb)
- cuDNN Code Samples and User Guide for Ubuntu18.04 (Deb)

在图形化界面下可以双击安装，也可以使用命令 `sudo dpkg -i 包名`

五、Tensorflow 的安装

1. 安装 tensorflow: `pip install tf-nightly-gpu`

Tensorflow 的官方网站上给出的安装教程中，软件的依赖如下图

Software requirements

The following NVIDIA® software must be installed on your system:

- [NVIDIA® GPU drivers](#) –CUDA 10.0 requires 410.x or higher.
- [CUDA® Toolkit](#) –TensorFlow supports CUDA 10.0 (TensorFlow >= 1.13.0)
- [CUPTI](#) ships with the CUDA Toolkit.
- [cuDNN SDK](#) (>= 7.4.1)
- (Optional) [TensorRT 5.0](#) to improve latency and throughput for inference on some models.

图 7-2 Tensorflow 依赖

2. 验证

```
python -c "from tensorflow.python.client import device_lib;  
print(device_lib.list_local_devices())" e
```

如果不提示错误，则安装成功。

六、Bert 的安装

1. 下载

1) 下载：源码 <https://github.com/google-research/bert/archive/master.zip>

预训练 模型：https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip

2. 预训练模型解压到桌面，设置环境变量

```
export BERT_BASE_DIR=~/Desktop/uncased_L-12_H-768_A-12
```

解压源码到桌面

7.2 准备数据

SQuAD 数据集有两个版本 1.1 和 2.0 的，可以下载 2.0 版本。

1. 下载数据

到 SQuAD 官网上下载三个文件，放在桌面上，

- train-v2.0.json
- dev-v2.0.json
- evaluate-v2.0.py

设置环境变量：

```
export SQUAD_DIR=~/Desktop
```

2. 数据内容：

SQuAD2.0 问答数据集并不复杂，训练集中包含了 442 段话题，然后每个话题有一个 title，对于每个话题下都有几十段的 paragraph，每个 paragraph 是一个 json 字典文件，包含了 context 和至少一组问答对：

```
input_data = {list} <class 'list'>: <Too big to print. Len: 442>
000 = {dict} <class 'dict'>: {'title': 'Beyoncé', 'paragraphs': [{'qas': [{'question': 'When did Beyonce start becoming popular?', 'id': '56be85543ae...'}]}]}
001 = {dict} <class 'dict'>: {'title': 'Frédéric Chopin', 'paragraphs': [{'qas': [{'question': 'What was Frédéric's nationalities?', 'id': '56cbd2356d24...'}]}]}
002 = {dict} <class 'dict'>: {'title': 'Sino-Tibetan relations during the Ming dynasty', 'paragraphs': [{'qas': [{'question': 'Who were Wang Jiawei...'}]}]}
003 = {dict} <class 'dict'>: {'title': 'iPod', 'paragraphs': [{'qas': [{'question': 'Which company produces the iPod?', 'id': '56cc55856d243a140015e...'}]}]}
004 = {dict} <class 'dict'>: {'title': 'The Legend of Zelda: Twilight Princess', 'paragraphs': [{'qas': [{'question': 'What category of game is Leger...'}]}]}
005 = {dict} <class 'dict'>: {'title': 'Spectre (2015 film)', 'paragraphs': [{'qas': [{'question': 'Which company made Spectre?', 'id': '56cdf0362d2...'}]}]}
006 = {dict} <class 'dict'>: {'title': '2008 Sichuan earthquake', 'paragraphs': [{'qas': [{'question': 'In what year did the earthquake in Sichuan oc...'}]}]}
007 = {dict} <class 'dict'>: {'title': 'New York City', 'paragraphs': [{'qas': [{'question': 'What city in the United States has the highest population...'}]}]}
008 = {dict} <class 'dict'>: {'title': 'To Kill a Mockingbird', 'paragraphs': [{'qas': [{'question': 'When did To Kill a Mockingbird first get circulated...'}]}]}
```

图 7-3 SQuAD 数据集示意图 1

这里 text 就是答案，answer_start:是指答案在文章开始的地方

```
'qas' (140365976924600) = {list} <class 'list'>: [{'question': 'When did Beyonce start becoming popular?', 'id': '56be85543aeaaa14008c9...'}]
00 = {dict} <class 'dict'>: {'question': 'When did Beyonce start becoming popular?', 'id': '56be85543aeaaa14008c9063', 'answers': [{'t...'}]}
question (140365976836272) = {str} 'When did Beyonce start becoming popular?'
id (140365976924936) = {str} '56be85543aeaaa14008c9063'
answers (140365976925048) = {list} <class 'list'>: [{'text': 'in the late 1990s', 'answer_start': 269}]
0 = {dict} <class 'dict'>: {'text': 'in the late 1990s', 'answer_start': 269}
__len__ = {int} 1
'is_impossible' (140365976900720) = {bool} False
__len__ = {int} 4
```

图 7-3 SQuAD 数据集示意图 1

7.3 训练模型

1. 训练方式

进入桌面上的 Bert 源码文件夹，在命令行中输入下面的命令

```
python run_squad.py \
--vocab_file=$BERT_BASE_DIR/vocab.txt \
--bert_config_file=$BERT_BASE_DIR/bert_config.json \
--init_checkpoint=$BERT_BASE_DIR/bert_model.ckpt \
--do_train=True \
--train_file=$SQUAD_DIR/train-v2.0.json \
--do_predict=True \
--predict_file=$SQUAD_DIR/dev-v2.0.json \
--train_batch_size=12 \
--learning_rate=3e-5 \
--num_train_epochs=2.0 \
--max_seq_length=384 \
--doc_stride=128 \
--version_2_with_negative=True \
--output_dir=/tmp/squad_base/
```

表 3- 1 Bert 训练方式

训练方式	训练时间 指标	准确率指标
BERT-Base, Uncased（小模型）	短	较高
BERT-Large, Uncased（大模型）	长	较高

7.4 测试

1. 运行 evaluate-v1.1.py

进入 output-dir 文件夹，有 predictions.json 文件，这是对 dev-v2.0.json 的预测

2. 运行测试命令：`python $SQUAD_DIR/evaluate-v2.0.py $SQUAD_DIR/dev-v2.0.json ./predictions.json`
3. 结果

表 3- 5 模型测试得分

训练方式	EM	F1
Human Performance	82.304	91.221
BERT-Base, Uncased (single model)	80.343	83.243
Tree-LSTM + BiDAF + ELMo (single model)	57.707	62.341

```
{"f1": 83.24349612335034, "exact_match": 80.3438174077578}
```

由于使用的是小模型，效果没有大模型好，但是还是很不错的。在 2018 年 10 月前，很少有模型的 f1 得分超过 85。在 BERT 开源后的半年时间中，SQuAD2.0 的数据集模型排名发生了很大变化。现在 2019 年 5 月，三名均使用了 BERT，前十名有 8 名使用了 BERT，前二十名中有 17 名使用了 BERT。传统 RNN 中 LSTM 已经没有在榜单上出现，只有 LSTM 的变种，与卷积神经网络结合的 ConvLSTM 出现在榜单中的 ensemble 模型中。使用 ELMo 的模型排名为 58/60。ELMo 的论文发表于 2018 年上半年，当 ELMo 源代码的预训练开源后，没过数月 BERT 的论文就发表了，几乎同时 Google 就在 Github 上开源了 BERT 源代码和预训练。现在第一名的 BERT 模型已经超过人类表现。

第8章 结论与展望

8.1 总结

本人的基于 SQuAD 数据集的问题回答模型通过测试，可以达到预先提出的目标 F1 得分 70 以上。在这个过程中，我学习到了很多知识。深度神经网络是循

神经网络的基础，自然语言处理既可以使用 RNN，也可以使用 CNN，针对一般 RNN 学习跨度较远的语法能力差的缺陷，研究者提出了 GRU, LSTM 算法。而数年前 Google 里有些人认为 RNN 有其固有的缺点，他们发表了 Attention is All You Need 论文，提出了 Transformer 算法，进而开发了 BERT 模型。我在这 3 个月中不断学习 DNN、RNN、CNN、NLP 相关的知识，收获了很多。

本文的研究灵感来源于斯坦福大学的 CS224n 课程，该课程的课程设计就是做一个自然语言模型。深度学习有三个应用场景：图像识别、自然语言处理、推荐系统。近几年，自然语言处理领域中几乎每年都有一到两个突破性的进展。但自然语言处理的发展没有图像识别快，直到 2018 年才有了有效的迁移学习。2018 年是自然语言处理领域重要的一年。

本文主要研究内容如下：

- 1) 神经网络和监督学习的关系、深度学习为什么发展这么快。逻辑回归和深度神经网络的基本算法：线性变换、sigmoid 非线性变换，ReLU 非线性变换、损失函数、代价函数、向前传播、向后传播、梯度下降、并给出使用 Python 中 Numpy 搭建的深度神经网络模型。
- 2) 深度神经网络的各种优化算法。主要包括 Mini-batch，避免过拟合 L2 正则化、Dropout，避免梯度消失和梯度爆炸的参数初始化。常用的三种梯度下降算法：Momentum、RmSprop、Adam 的公式推导和理解。
- 3) 近数年来，循环神经网络和卷积神经网络的发展历史，各个时期的热门算法和模型的优点和不足。Word embedding、Word2Vec、GRU、LSTM、Transformer、BERT 相关公式的推导，模型的架构。
- 4) 深度学习方面的环境搭建，主要是 Ubuntu 操作系统的安装、Ubuntu 版本和 Python 版本的关系、显卡驱动和 Cuda 的安装、和 Tensorflow 的安装。通过这些环境的搭建，可以增加计算机操作系统理解，以及实际动手能力，耐心和对硬件性能的理解。
- 5) 怎么使用开源 BERT 模型来训练数据集，如何发挥电脑性能。未来 NLP 和 BERT 框架可能的发展方向。

8.2 展望

本人使用 Google 开源的 BERT 模型，和 Google 使用大量训练样本花费四天时间在 10 个 TPU 上训练得到的预训练网络参数，完成了对 SQuAD 数据集的测试。测试结果满足预期要求，但还存在这可以改进的地方。今后对研究工作的展望如下：

- 1) 由于我现在电脑的硬件并不是特别好，使用的 BERT 网络是小网络，参数数量为 1 亿 1 千万，而大网络的参数数量是 3 亿 4 千万，使用大网络会有更好的效果。
- 2) 我使用 SQuAD 训练集对 BERT 模型的微调（fine-tune）一个多小时后，发现其效果并没有比不微调有明显的提升。可能应该让模型微调一两天才有效果。
- 3) 现在我是用的是单模型，也就是只有 BERT 模型。一般使用多模型（ensemble）可以有 5% 以内的提升。
- 4) 随着自然语言处理领域的快速发展，本项目移据 BERT 提供的源码在 Tensorflow 框架训练的模型也有可能被更好的方法替代，在 SQuAD 数据集的排行榜上，有可能出现其他比 BERT 更好的模型。因此，要跟着自然语言处理前沿不发，不断了解前沿技术。

—

致 谢

光阴似箭，日月如梭，我的大学本科时光就要结束，在这段日子里，我收获到了知识的快乐、老师的帮助、朋友的关心和同学的友情。

首先，我要感谢上海大学自动化系的周维民老师，因为在他的悉心教导下，我很好地完成了毕业项目的设计以及毕业论文的撰写。我在毕业设计期间，遇到了许多经过很长时间而无法解决的难题，周老师不论工作有多繁忙，总是能抽出时间帮我整理思路、指引方向，为我提供了一些改善性的帮助，并且对我的研究方向做出了许多建设性的指导意见，使我从中获益匪浅。在此，我诚挚地向周老师表达深深的敬意和由衷的感谢。

我还要感谢在生活上给予我支持、在精神上给予我鼓励的父母，是他们辛勤的哺育、无私的奉献给了我巨大的信心和力量，使我能在学校全心全意地投入到知识的海洋里，认真仔细、刻苦钻研。

最后，感谢参与论文评阅、指导的专家们！感谢所有陪伴我、关心我、帮助过我的人们！

参考文献

- [1] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[J]. 2014:1-9.
- [2] Deng J, Dong W, Socher R, et al. ImageNet: A large-scale hierarchical image database[J]. Proc of IEEE Computer Vision & Pattern Recognition, 2009:248-255.
- [3] ImageNet. Large Scale Visual Recognition Challenge 2014(ILSVRC2014) [EB/OL]. <http://image-net.org/challenges/LSVRC/>.
- [4] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[C]// Computer Vision and Pattern Recognition. IEEE, 2016:770-778.
- [5] 赵永科. 深度学习:21 天实战 Caffe[M]. 北京:电子工业出版社, 2016: 50-78,82-99,101-139,193-227,275.
- [6] 乐毅 . 深度学习—— Caffe 之经典模型详解与实战 [M].北京 : 电子工业出版社 .2016
- [7] 杨云, 杜飞 .深度学习实战 [M].北京: 清华大学出版社.2018.
- [8] 袁国忠. Python 编程: 从入门到实践[M].北京: 人民邮电出版社 .2016
- [9] 用数据做酷的事! 手把手教你搭建问答系统 [EB/OL] <https://zhuanlan.zhihu.com/p/35340535>
- [10] Chris Manning. CS 224N Default Final Project: Question Answering on SQuAD 2.0 <http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf> 2018
- [11] Jacob Devlin. BERT <https://github.com/google-research/bert> 2018.
- [12] Ashish Vaswani, Noam Shazeer: Attention Is All You Need, 2017:1-12
- [13] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Efficient Estimation of Word Representations in Vector Space 2013:1-16
- [14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, Distributed Representations of Words and Phrases and their

Compositionality.2013

[15] 李金洪 深度学习之 Tensorflow 入门、原理与进阶实战： 2018