# 计算机博士半小时【Transformer入门实战

【导师不教？我来教！】计算机博士半小时就教会了我【Transformer入门实战】教程！草履虫都能学会！

4953 2022-11-09 12:12:09 [小叮当叮咚叮](https://space.bilibili.com/411681414" \t "https://www.bilibili.com/video/_blank)

https://www.bilibili.com/video/BV1Lg411z7gZ?p=1

黑马程序员 www.itheima.com

# P1 1.1.Transformer 05:42

## 第一章:Transformer背景介绍

### 1.1 Transformer的诞生

2018年10月，Google发出一论文《BERT:Pre-training of Deep Bidirectional Transformers for

Language Understanding》,BERT模型模空出世,并机扫NLP领域11项任务的最佳成绩!

论文地址: https://arxiv.org/pdf/1810.04805.pdf

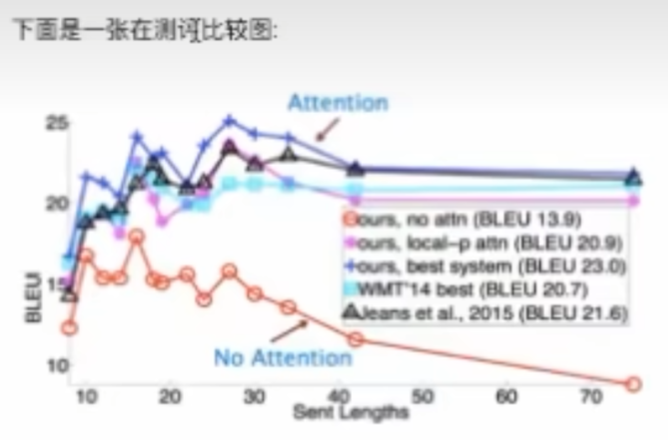
而在BERT中发挥常要作用的结构就是Transformer,之后又相维出现XLNET，roBERT等模型击败了BERT，但是他们的核心没有变，仍然是:Transformer.

### 1.2 Transformer的优势

相比之前占领市场的LSTM和GRU模型。Transformer有两个显著的优势:

1。Transforaer能够利用分布式GPU进行并行训练，提升模型训练效率

2。在分拆预期更长的文本时，捕捉间隔较长的语义关联效果更好。



### 1.3 Transformer的市场

在著名的SOTA机器翻译榜单上,几乎所有排名靠前的模型都使用Transformer。

其基本上可以看作是工业界的风向标,市场空间自然不必多说！

# P2 2.1认识Transformer架构-part1 07:57

## 第二章:Transformer架构解析

### 2.1 认识Transformer架构

学习目标:

。了解Transformer模型的作用

·了解Transformer总体架构图中各个组成部分的名称

Transformer模型的作用:

基于seq2seq架构的transformer模型可以完成NLP领域研究的典型任务,如机器翻译,文本

生成等.同时又可以构建预训练语言模型，用于不同任务的迁移学习

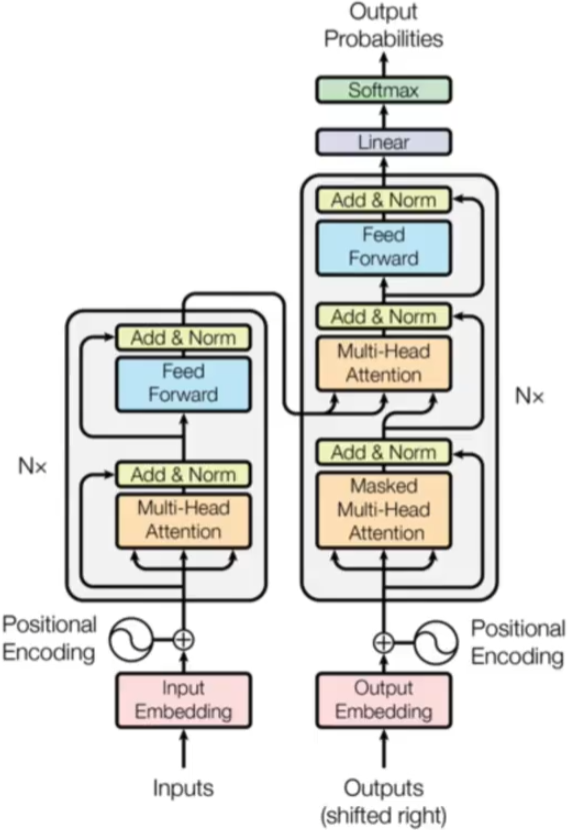
声明:

在接下来的架构分析中,我们将假设使用Transformer模型架构处理从一种语言文本到另一

种语言文本的**翻译**工作,因此很多**命名方式遵循NLP中的规则**.比如:Embeddding层将称作

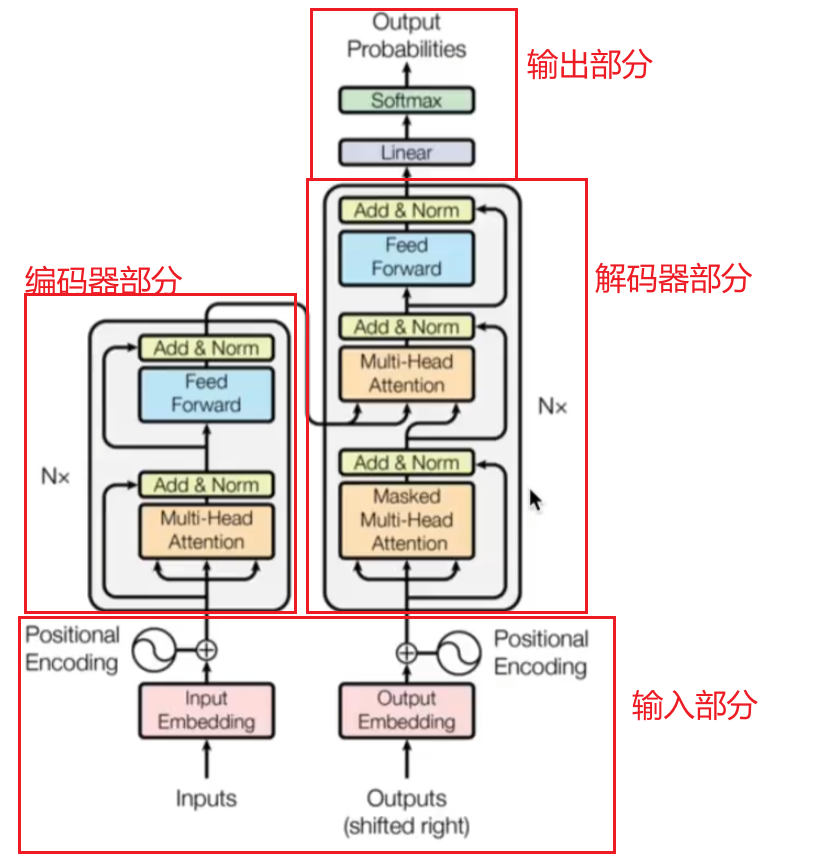
文本嵌入层,Embedding层产生的张量称为词嵌入张量,它的最后一维将称作词向量等

Transformer总体架构图



。Transformer总体架构可分为四个部分:（1）输入部分（2）输出部分（3）编码器部分

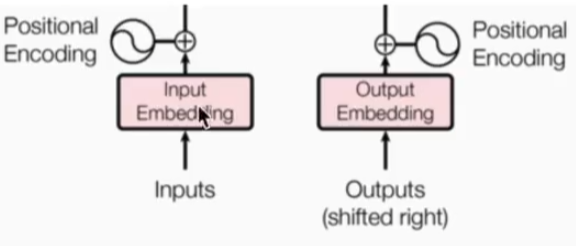
解码器部分



·输入部分包含:

\*源文本嵌入层 及其位置编码器

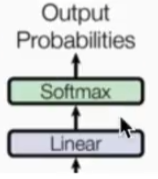
\*目标文本嵌入层 及其位置编码器



输出部分包含:

\*线性层

\*softmax层



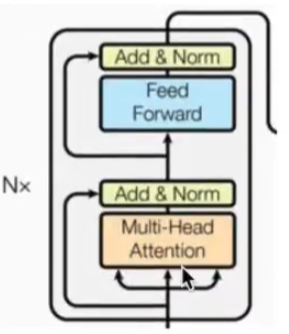
编码器部分:

\*由N个编码器层堆叠而成

\*每个编码器层由两个子层连接结构组成

\*第一个子层连接结构包括一个多头自注意力子层和规范化层以及一个残差连接

\*第二个子层连接结构包括一个前馈全连接子层和规范化层以及一个残差连接



解码器部分:

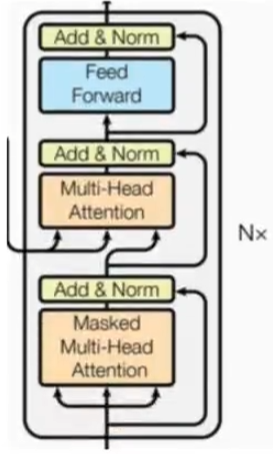
\*由N个解码器层堆叠而成

\*每个解码器层由三个子层连接结构组成

\*第一个子层连接结构包括一个多头自注意力子层和规范化层以及一个残差连接

\*第二个子层连接结构包括一个多头注意力子层和规范化层以及一个残差连接

\*第三个子层连接结构包括一个前馈全连接子层和规范化层以及一个残差连接



# P3 2.1认识Transformer架构-part2

04:10

对2.1小节的总结（略）

# P4 2.2输入部分实现-part1-05:04

## 2.2 输入部分实现

学习目标:

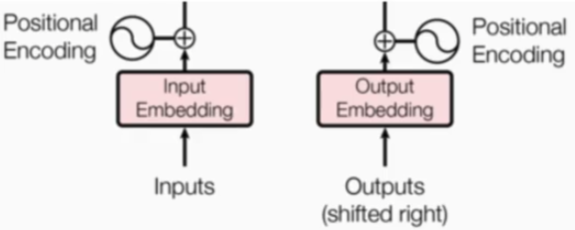
\*了解文本嵌入层和位置编码的作用

\*掌握文本嵌入层和位置编码的实现过程

输入部分包含

·源文本嵌入层及其位置编码器

·目标文本嵌入层及其位置编码器



文本嵌入层的作用:无论是源文本嵌入还是目标文本嵌入，都是为了将文本中词汇的数字表示转变为向量表示，希望在这样的**高维空间**捕捉词汇间的关系

**pytorch 0.3.0及其必备工具包的安装：**



# 使用pip安装的工具包包括pytorch-0.3.0，numpy，matplotlib，seaborn

pip install http://download.pytorch.org/whl/cu80/torch-0.3.0.post4-cp36-cp36m-linux

#MAC系统安装，python版本<=3.6

pip install torch==0.3.0.post4 numpy matplotlib seaborn

文本嵌入层的代码分析:

####################################

# 导入必备的工具包

import torch

#预定义的网络层torch.nn，工具开发者已经帮助我们开发好的一些常用层

#比如，卷积层，lstm层，embedding层等，不需要我们再重新造轮子

import torch.nn as nn

# 数学计算工具包

import math

# torch中变量封装函数Variable.

from torch.autograd import Variable

# 定义Embeddings类来实现文本嵌入层，这里s说明代表两个一模一样的嵌入层，他们共享参数

# 该类继承nn.Module，这样就有标准层的一些功能，这里我们也可以理解为一种模式，我们自己实现的所有层

class Embeddings(nn.Module):

    def \_\_init\_\_(self,d\_model,vocab):

    """类的初始化函数,有两个参数,d\_model:指词嵌入的维度，vocab:指词表的大小."""

    # 接着就是使用super的方式指明继承nn.Module的初始化函数，我们自己实现的所有层都会这样去

    super(Embeddings,self).\_\_init\_\_()

    #之后就是调用nn中的预定义层Embedding，获得一个词嵌入对象self.lut

    self.lut =nn.Embedding(vocab,d\_model)

    # 最后就是将d\_model传入类中

    self.d model=d model

    def forward(self,x):

        """可以将其理解为该层的前向传播逻辑，所有层中都会有此函数

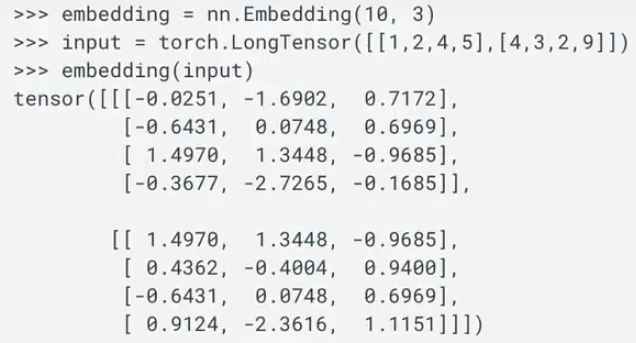
        当传给该类的实例化对象参数时，自动调用该类函数

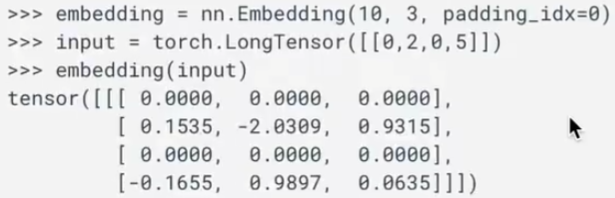
        参数x:因为Embedding层是首层，所以代表输入给模型的文本通过词汇映射后的张量 """

        # 将x传给self.lut并与根号下self.d\_model相乘作为结果返回

        return self.lut(x)\*math.sqrt(self.d\_model)

nn.Embedding演示:





# P5 2.2输入部分实现-part2 -09:36

# P6 2.2输入部分实现-part3-13:14

# P7 2.2输入部分实现-part4 -11:36

# P8 2.2输入部分实现-part5 -06:15

# P9 2.2输入部分实现-part6-04:29

P10

10. 2.3.1掩码张量-part1

07:09

P11

11. 2.3.1掩码张量-part2

07:44

P12

12. 2.3.1掩码张量-part3

03:17

P13

13. 2.3.2注意力机制-part1

09:17

P14

14. 2.3.2注意力机制-part2

06:08

P15

15. 2.3.2注意力机制-part3

10:50

P16

16. 2.3.2注意力机制-part4

04:13

P17

17. 2.3.3多头注意力机制-part1

19:28

P18

18. 2.3.3多头注意力机制-part2

07:04

P19

19. 2.3.3多头注意力机制-part3

16:27

P20

20. 2.3.3多头注意力机制-part4

06:17

P21

21. 2.3.4前馈全连接层-part1

05:53

P22

22. 2.3.4前馈全连接层-part2

08:45

P23

23. 2.3.5规范化层-part1

06:06

P24

24. 2.3.5规范化层-part2

08:56

P25

25. 2.3.6子层连接结构-part1

05:36

P26

26. 2.3.6子层连接结构-part2

08:51

P27

27. 2.3.7编码器层-part1

06:01

P28

28. 2.3.7编码器层-part2

10:22

P29

29. 2.3.8编码器-part1

04:17

P30

30. 2.3.8编码器-part2

08:38

P31

31. 2.4.1解码器层-part1

13:16

P32

32. 2.4.1解码器层-part2

15:16

P33

33. 2.4.2解码器-part1

04:41

P34

34. 2.4.2解码器-part2

10:06

P35

35. 2.5输出部分实现-part1

04:55

P36

36. 2.5输出部分实现-part2

06:21

P37

37. 2.6模型构建-part1

08:20

P38

38. 2.6模型构建-part2

11:37

P39

39. 2.6模型构建-part3

07:48

P40

40. 2.6模型构建-part4

15:11

P41

41. 2.7模型基本测试运行-第二步-part1

09:06

P42

42. 2.7模型基本测试运行-第二步-part2

07:29

P43

43. 2.7模型基本测试运行-第三步

08:25

P44

44. 2.7模型基本测试运行-第四步

12:40

P45

45. 2.7模型基本测试运行-第一步-part1

08:37

P46

46. 2.7模型基本测试运行-第一步-part2

05:54

P47

47. 2.7模型基本测试运行-小节总结

09:59

P48

48. 3.1使用Transformer构建语言模型-0总体介绍

07:27

P49

49. 3.1使用Transformer构建语言模型-第1步

08:44

P50

50. 3.1使用Transformer构建语言模型-第2步-part1

06:26

P51

51. 3.1使用Transformer构建语言模型-第2步-part2

06:30

P52

52. 3.1使用Transformer构建语言模型-第3步-part1

17:42

P53

53. 3.1使用Transformer构建语言模型-第3步-part2

11:00

P54

54. 3.1使用Transformer构建语言模型-第4步-part1

08:42

P55

55. 3.1使用Transformer构建语言模型-第4步-part2

16:02

P56

56. 3.1使用Transformer构建语言模型-第4步-part3

07:48

P57

57. 3.1使用Transformer构建语言模型-第5步-part1

15:54

# P58 58. 3.1使用Transformer构建语言模型-第5步-part2

06:04

# **附1.GPT图解:大模型是怎样构建的 推荐学习**

序章 看似寻常最奇崛，成如容易却艰辛

第 1 课 高楼万丈平地起：语言模型的雏形 N-Gram 和简单文本表示 Bag-of-Words

第 2 课 问君文本何所似：词的向量表示 Word2Vec 和 Embedding

第 3 课 山重水复疑无路：神经概率语言模型和循环神经网络

第 4 课 柳暗花明又一村：Seq2Seq 编码器 - 解码器架构

第 5 课 见微知著开慧眼：引入注意力机制

第 6 课 层峦叠翠上青天：搭建 GPT 核心组件 Transformer

第 7 课 芳林新叶催陈叶：训练出你的简版生成式 GPT

第8课 流水后波推前波：ChatGPT基于人类反馈的强化学习

第9课 生生不息的循环：使用强大的GPT-4 API

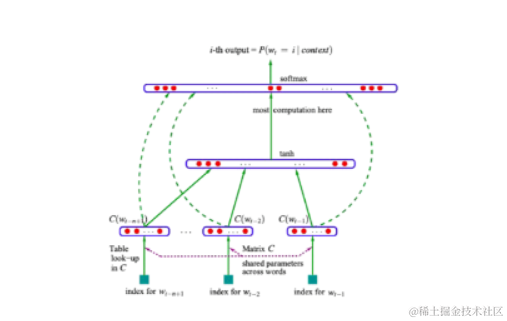
后 记 莫等闲，白了少年头

****项目一：N-Gram构建****

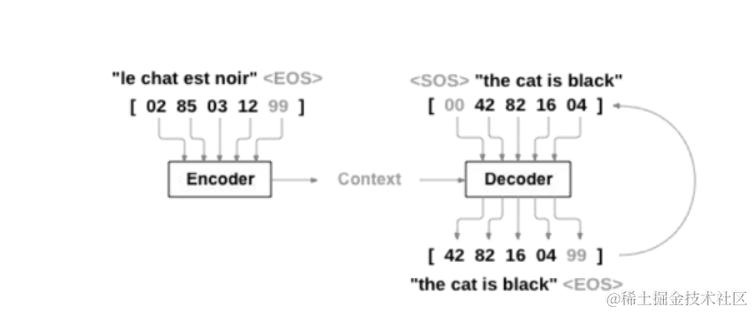


****项目二：Word2Vec构建****  

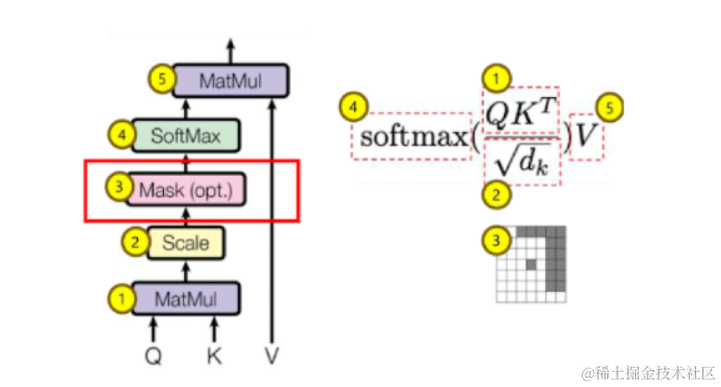

****项目三：NPLM（Neural Probabilistic Language Model）构建****



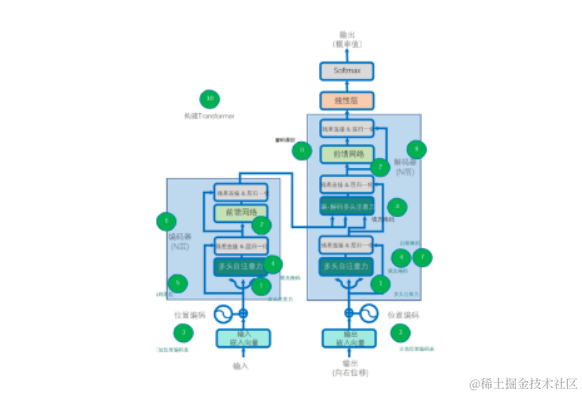
****项目四：Seq2Seq架构****



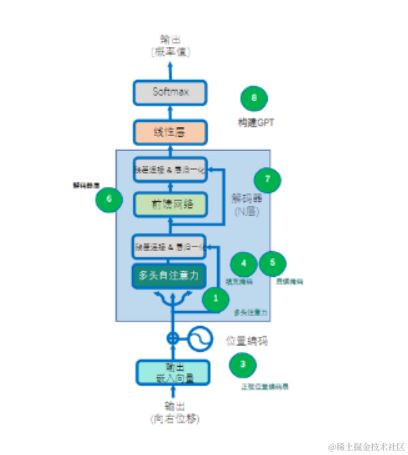
****项目五：注意力机制****



****项目六：Transformer架构****



****项目七：WikiGPT****



****项目八：miniChatGPT****

