揚州大學
YANGZHOU UNIVERSITY

自 然 语 言 处 理

Natural language Processing

基于深度学习框架的序列处理

第6章

CONTENTS

01 语言模型回顾

02 循环神经网络

03 LSTM

04 Transformer

语言的本质

语言本身是人类进化发展伴随而来的一种时间现象。

口语生成和变化是随着随着时间的推移的一系列声学时间。

口语和人类文字语言可以理解为一个连续输出流。

语言的时间性体现在：人类语言是随着时间展开的序列。

- flow of conversations,
- news feeds, and
- twitter streams

语言处理

语言的时间性质反映在用于处理语言的一些算法中。

- 例如：可用于隐式马尔科夫模型(HMM)词性标记的Viterbi算法
  - 一次输入一个单词，并且携带之前文本的信息

不具有时间性质的机器学习模型：

- 例如之前学习过的情感分析或者其他的文本分类任务
  - 因为这些模型把单词序列当做一个整体对待，
  - 即同时访问并处理输入的所有信息。

Part.01

语言模型回顾

回顾：前馈神经网络

上一章中讲到的前馈神经网络

- 假设对数据进行同时访问

- 尽管也具有一个有关时间的简单模型
  - 有固定大小的单词窗口，
  - 在输入上滑动窗口，
  - 在该过程中完成独立预测。

回顾：前馈神经网络

p(ant|...)
p(doe|...)
p(fish|...)
p(zebra|...)
output layer y
U
hidden layer h
h
W
embedding layer e
... and thanks for all the ? ...
w_{t-3}
w_{t-2}
w_{t-1}
w_t

简单的前馈神经网络适用于序列处理？

尽管先前介绍到的前馈神经网络，带有滑动窗口

- 可以处理该窗口内的单词预测任务。
- 但是并不完全令人满意，为什么？
  - 前馈网络相对于N-grams
    - 是从基于单词的计数统计转化为使用词向量(词嵌入)
      - 可以解决相似单词之间的泛化问题
    - 但两者有共同的劣势-有限的上下文
      - 前馈为滑动窗口
      - N-grams为n
  - 滑动窗口的使用使得网络很难学习到系统性模式
    - 比如语法成分，或者有含义的单词组合(短语)

滑动窗口的劣势

p(ant|...)
p(doe|...)
p(fish|...)
p(zebra|...)
output layer y
U
hidden layer h
h
W
embedding layer e
... and thanks for all the ? ...
w_{t-3}
w_{t-2}
w_{t-1}
w_t

本章内容概述

为了应对文本序列的时间属性，并挑战先前的语言模型劣势。

介绍两种深度学习架构：

- RNN循环神经网络(recurrent neural networks)
  - 提出一种新的方式来表示先前文本内容的信息
  - 当前的决策基于先前序列的信息，甚至几百个单词
- Transformer网络
  - 自注意力机制
  - 位置编码
  - 有助于时间表示和长距离单词之间的关联

本章内容概述

这两种架构都提供了处理语言序列属性的功能机制，

这两种模型将应用于

- 语言建模任务、
- 序列建模任务（如词性标记）
- 文本分类任务（如情感分析）。

概率语言模型用于预测序列中的下一个单词

举例：想要预测“Thanks for all the”之后，‘fish’作为下一个单词出现的概率，

需要对下面的概率进行计算

$$P(fish|Thanks\ for\ all\ the)$$

语言模型可以把这样的条件概率分配给每一个可能的单词，

- 即对整个词汇表提供一个概率分布

概率语言模型用于预测序列中的下一个单词

所有单词的条件概率可以与链式法则结合

- 为整个序列分配概率

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i | w_{<i})$$

怎样评估一个好的语言模型

模型预测未曾出现在训练集中的单词：

- 可以分配更高的概率给新词(对新单词的出现并不惊讶)

使用困惑度(perplexity)来评估模型好坏：

- 模型 θ 分配给测试数据的逆概率
- 根据测试数据的长度做归一化处理
- 对于一个测试数据$w_{1:n}$, 其困惑度为

$$\begin{aligned} \mathrm{PP}_\theta(w_{1:n}) &= P_\theta(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P_\theta(w_{1:n})}} \end{aligned}$$

困惑度 + 链式法则

困惑度计算

- 衡量语言模型在计算每个新单词概率的优劣
- 结合使用链式法则，扩展测试集概率的计算

$$\mathrm{PP}_\theta(w_{1:n}) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P_\theta(w_i | w_{1:n-1})}}$$

Part.02

RNN神经网络

什么是RNN循环神经网络

Recurrent Neural Networks 循环神经网络是指：

- 任何在其网络连接中包含循环的神经网络模型
  - 某一个输出值直接或间接的依赖于先前的输入
  - 功能强大
  - 难以推理和理解
  - 已经被证明其处理语言序列时的有效性。

- 基于RNN的模型包括Elman, LSTM等

简单的RNN神经网络模型

RNN循环神经网络

来自于前一个时间戳的隐藏层以一种上下文或者记忆的形式

- 对先前的序列处理进行编码
- 并将编码的信息提供给后续的决策制定。

注意！

- RNN网络没有对先前上下文的长度进行限制。
- 在前一个隐藏层的编码信息中包含着整个序列的第一个输入的信息。

RNN循环神经网络

RNN的时间性质，使得RNN比非循环结构模型更加复杂

但RNN也利用了前馈的机制。

权重矩阵U用于连接
当前和前一个隐藏层。

使用反向传播算法训练U

RNN分析

RNN的前向推理与前馈网络几乎一样

- 将输入序列映射到输出序列

RNN输入到输出的映射过程需要隐藏层的h_t的激活值。

- 首先将输入x_t
  1. **乘以**权重矩阵W
  2. 将结果**加上**第t-1步隐藏层h_{t-1}的加权向量$U*x_{t-1}$
- 然后使用激活函数g对上一步计算的结果进行非线性转换
- 在将结果送入当前隐藏层h_t
- 最后可以使用h_t来正常计算当前的输出向量。

RNN分析

输入层，隐藏层，输出层的维度分别为d_{in}, d_h, d_{out}

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

$W \in \mathbb{R}^{d_h \times d_{in}}$ $\mathbf{U} \in \mathbb{R}^{d_h \times d_h}$ $\mathbf{V} \in \mathbb{R}^{d_{out} \times d_h}$

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{V}\mathbf{h}_t)$$

RNN分析

在时间t上的计算需要时间t-1时间上隐藏层的值
需要一个增量式推理算法来处理序列的开始到结束。

function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

$\mathbf{h}^0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

RNN网络展开

U,V,W跨时间共享

每一步都会计算h_i, y_i

RNN模型训练

与前馈网络训练一样，需要使用训练集，损失函数和反向传播算法来计算梯度

- 梯度可以用于权重矩阵的调整和更新
- 有三个权重矩阵(U,V,W)

RNN模型训练

计算时间t时输出的损失函数需要时间t-1的隐藏层信息。

时间t的隐藏层会影响

- 当前输出
- 时间t+1的隐藏层、输出和损失
- 因此需要评估h_t产生的误差

RNN模型训练

需要对反向传播进行调整来应对这种情况，需要分两步:

1. 向前推理，计算h_t，yt, 在每一时间步上累积损失，
并且保存隐藏层的值以供下一时间步上使用

2. 对序列反向处理，计算所需的梯度，
并且保存误差项，以便在每次后退时用于隐藏层

这种方法称为随时间反向传播(Backpropagation Through Time)

RNN语言模型

有一个文本序列X包含N个单词向量。 $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_t; \ldots; \mathbf{x}_N]$
每一个单词x_t用one-hot encoding去表示。向量维度为|V|，V是词汇表大小。
通过词向量矩阵，可以获取单词x_t对应的词向量。

输出预测 y 是一个|V|维向量，表示所有词汇表上单词出现在当前位置的概率分布。

在时间t上：

$$
\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\mathbf{y}_t &= \mathrm{softmax}(\mathbf{V}\mathbf{h}_t)
\end{aligned}
$$

W 是输入层到当前隐藏层权重
U是从前一隐藏层到当前隐藏层权重
V是从当前隐藏层到输出层权重
E是词嵌入矩阵

RNN语言模型

在当前预测中，词汇表中的单词i出现在下一个位置的概率可以表达为：

$$P(w_{t+1} = i | w_1, \ldots, w_t) \;=\; \mathbf{y}_t[i]$$

$y_t[i]$是输出向量y_t的第i个元素。

那么整个序列的概率，可以表达为每一位置单词出现概率的乘积。

$$\begin{aligned} P(w_{1:n}) &= \prod_{i=1}^{n} P(w_i | w_{1:i-1}) \\ &= \prod_{i=1}^{n} \mathbf{y}_i[w_i] \end{aligned}$$

多分类损失函数

二分类损失函数

$$L_{CE}(\hat{y},y) = -\log p(y|x) \;=\; -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$

多分类损失函数

$$L_{CE}(\hat{\mathbf{y}},\mathbf{y}) = -\sum_{k=1}^{K}\mathbf{y}_k \log\hat{\mathbf{y}}_k$$

使用损失函数计算预测分布与真实分布的误差

交叉熵(cross entopy)

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

交叉熵损失是由预测模型分配给正确下一个单词的概率决定的。
那么在时间t，交叉熵损失可以表示为：

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

为什么？

教师强制

然而在每一步预测的单词都有可能是错误的该如何解决？

在预测x_{t+1}时，使用真实单词序列$w_{1:t}$，而不使用预测出来的单词序列

这种方法称为教师强制

使用RNN作为语言模型

所有项交叉熵的平均值用以估计损失
(预测值与目标值的差值)

$$\frac{1}{T}\sum_{t=1}^{T} L_{CE}$$

使用RNN作为语言模型

输入向量矩阵E 与 可以作为softmax函数输入的最终层矩阵V很相似

- E的每一行对于的词汇表相应单词的词向量,
- 这些词向量是在训练过程中计算得到的。

矩阵E的shape为|V| x d_h,

- 因为这些向量的维度等于隐藏层神经元的个数(d_h)

$$
\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\mathbf{y}_t &= \mathrm{softmax}(\mathbf{E}^{intercal}\mathbf{h}_t)
\end{aligned}
$$

通过计算Vh, 最终层矩阵V提供了一种对词汇表中

- 每一个单词的可能性进行评分的方法
- 也就是说计算得到的词向量具有计算得到的词义和单词功能的性质。

权重共享

E中包含了v个词向量

V中学习到了v个词向量

是否需要同时保留这两种权重矩阵？

$$
\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\mathbf{y}_t &= \text{softmax}(\mathbf{E}^{intercal}\mathbf{h}_t)
\end{aligned}
$$

可以提升模型的预测性能
也可以减少需要训练的参数量

RNN用于其他NLP任务

序列标注(sequence labelling)

序列分类(sequence classification)

文本生成(text generation)

序列标注(sequence labelling)

带上下文信息的多分类任务

分类项个数为标签集的大小

序列标记(sequence labelling)

$$\frac{1}{T}\sum_{t=1}^{T} L_{CE}$$

T个时间步

序列分类

情感分类(正面/负面)

主题分类

垃圾邮件检测

等等

序列分类

通过softmax计算所有分类项的可能性

预测值与真实值做交叉熵计算损失

将计算得到的损失逐层反向传播到所有权重矩阵 (W,U,V)， 并将其元素进行逐个计算修改。

序列分类

池化函数

$$\mathbf{h}_{mean} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i$$

对所有时间步产生的隐藏层状态，求平均

序列分类

在所有时间步上的隐藏层向量中h_i的第K个元素最大值

基于RNN文本生成

问答系统(question answering)

机器翻译(machine translation)

文本摘要(text summarization)

对话系统(conversational dialogue)

...

等等需要自动生成文本的任务。

回顾N-grams 文本生成

1. 首先根据单词的合适程度，随机选择一个单词作为文本开头，

2. 然后根据已选择的单词，以及N-grams语法生成后续的单词，

3. 直到达到预设的文本长度，或者直到生成预设的序列结束词(token)。

自回归生成(autoregressive generation)

1. 使用<s>作为句子第一个输入的token,
2. 基于当前的输入<s>，从t_0时刻的softmax分布中采样一个单词,
3. 使用第一个单词向量作为下一个时间步神经网络的输入,
   - 并且以同样的方式生成下一个单词,
4. 继续生成，直到生成句尾标注</s>，或者达到预设长度。

自回归生成(autoregressive generation)

隐藏层和循环连接

t_0 t_1 t_2 t_3 ……

堆叠(Stacked) RNNs

双向(Bidirectional) RNNs

序列是有方向的，之前介绍的RNN中，ht携带的是xt左侧的信息

$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

为了获取当前输入右侧的信息，可以将序列反转，再次训练一个RNN模型

$$\mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

将上述两种训练的隐藏层向量结合为一个新的向量，使用⊕表示连接关系

$$\begin{aligned} \mathbf{h}_t &= [\mathbf{h}_t^f ; \mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b \end{aligned}$$

双向(Bidirectional) RNNs

双向(Bidirectional) RNNs

Simple RNN for IMDB Review

```
from keras.datasets import imdb
import numpy as np


(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=500)
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

x_val = x_train[:3000]
x_train = x_train[3000:]
y_val = y_train[:3000]
y_train = y_train[3000:]
```

Simple RNN for IMDB Review

```
def vectorize_sequences(sequences, dimension=500):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

Simple RNN for IMDB Review

```
from keras.models import Sequential
from keras.layers import  SimpleRNN, Embedding, Dense

vocabulary = 10000
embedding_dim = 32
word_num = 500
state_dim = 32

model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(SimpleRNN(state_dim,return_sequences=False))
model.add(Dense(1,activation='sigmoid’))

model.summary()
```

Simple RNN for IMDB Review

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 32)           320000

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
```

Simple RNN for IMDB Review

```
from keras import optimizers

epochs = 3
model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=epochs, batch_size=512, validation_data=(x_val,
y_val))
```

```
Epoch 1/3
43/43 [==============================] - 6s 124ms/step - loss: 0.6807 -
acc: 0.5627 - val_loss: 0.6713 - val_acc: 0.5813
Epoch 2/3
43/43 [==============================] - 5s 123ms/step - loss: 0.6706 -
acc: 0.5846 - val_loss: 0.6821 - val_acc: 0.5497
Epoch 3/3
43/43 [==============================] - 5s 124ms/step - loss: 0.6686 -
acc: 0.5919 - val_loss: 0.6638 - val_acc: 0.6013
782/782 [==============================] - 5s 7ms/step - loss: 0.6623 -
acc: 0.6057
_________________________________________________________________________________
____________________________
Test Loss and Accuracy
results  [0.6623082756996155, 0.605679988861084]
```

Simple RNN for IMDB Review

```
from keras import optimizers

epochs = 3
model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
          loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=epochs, batch_size=512, validation_data=(x_val,
y_val))
```

```
Epoch 1/3
43/43 [==============================] - 6s 124ms/step - loss: 0.6807 -
acc: 0.5627 - val_loss: 0.6713 - val_acc: 0.5813
Epoch 2/3
43/43 [==============================] - 5s 123ms/step - loss: 0.6706 -
acc: 0.5846 - val_loss: 0.6821 - val_acc: 0.5497
Epoch 3/3
43/43 [==============================] - 5s 124ms/step - loss: 0.6686 -
acc: 0.5919 - val_loss: 0.6638 - val_acc: 0.6013
782/782 [==============================] - 5s 7ms/step - loss: 0.6623 -
acc: 0.6057
```

Simple RNN for IMDB Review

```
results = model.evaluate(x_test, y_test)
print("Test Loss and Accuracy")
print("results ", results)
```

```
Test Loss and Accuracy
results  [0.6623082756996155,
```

$\mathrm{sigmoid}(\mathbf{v}^T\ \mathbf{h})$
Flatten: $\mathbf{h} = \mathrm{vec}([\mathbf{h}_1, \cdots, \mathbf{h}_t])$
h_0
h_1
h_2
…
h_t
A
A
A
A
x_0
x_1
x_2
…
x_t
i
love
the
…
much

Simple RNN for IMDB Review

```
from keras.layers import Flatten

model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

Simple RNN for IMDB Review

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 32)           320000

 simple_rnn (SimpleRNN)      (None, 500, 32)           2080

 flatten (Flatten)           (None, 16000)             0

 dense (Dense)               (None, 1)                 16001

=================================================================
Total params: 338,081
Trainable params: 338,081
```

```
_____________________________________
Test Loss and Accuracy
results  [0.39525976777707672, 0.8226400017738342]
```

Stacked RNN for IMDB Review

```
model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

双向 RNN for IMDB Review

```
from keras.layers import Bidirectional

model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(Bidirectional(SimpleRNN(state_dim,return_sequences=True)))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

```
_____________________________
Test Loss and Accuracy
results  [0.39559492468833923, 0.8216800093650818]
```

Part.03

LSTM

RNN缺点

RNN劣势：不适合长序列文本处理

- 尽管可以访问整个序列，但隐藏状态中的编码信息是非常局部的
- 当前状态与最近的前序输入更相关
- 长距离信息在许多NLP任务重很重要

The flights the airline was cancelling were full.

原因：

- 隐藏层权重用于 1)为当前决策提供信息 2) 为未来决策更新信息。
- 梯度消失(vanishing gradients)

LSTM长短期记忆 (Long short-term memory)

Hochreiter and Schmidhuber, 1997

通过设计gate门机制实现,

从上下文中删除不需要的信息

添加对以后决策有帮助的信息

LSTM长短期记忆 (Long short-term memory)

有那些门？

遗忘门(forget gate), 输入门(input gate), 输出门(output gate)

遗忘门(forget gate)

作用是为了从上下文中删除不需要的信息

1. 计算前一个隐藏层与当前输入的加权和
2. 用激活函数sigmoid非线性变换
3. 将结果与C_{t-1}做阿达玛乘积(Hadamard product)
4. 删除上下文中不需要的信息

$$\mathbf{f}_t = \sigma(\mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t)$$
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$

输入门(input gate)

g_t携带之前状态信息和当前输入信息

$$\mathbf{g}_t = \tanh(\mathbf{U}_g\mathbf{h}_{t-1} + \mathbf{W}_g\mathbf{x}_t)$$

$$\mathbf{i}_t = \sigma(\mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t)$$

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

C_t成为下一时间步的上下文向量

$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$

输出门(output gate)

决定当前隐藏层需要那些信息(不考虑以后)

$$\mathbf{o}_t = \sigma(\mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LSTM 结构图

t-1时上下文向量

t-1时隐藏层状态

当前输入

更新后的

t时上下文向量

t时隐藏层状态

σ Sigmoid 函数

tanh 双曲正切函数

⊙ 阿达玛乘积

蓝色区域代表不同权重矩阵

对比 - 神经元, SRNN, LSTM

前馈网络机制

LSTM for IMDB Review

```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(LSTM(state_dim, return_sequences=True))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

```
_________________________________
Test Loss and Accuracy
results  [0.38315340876579285, 0.8312399983406067]
```

Part.04

Transformer

自注意力网络Transformer

LSTM 虽然可以保留长距离信息，但不能解决根本问题

- 通过多重循环连接传递信息会导致信息丢失
- 训练困难
- 不容易实现并行(顺序特性)

Transformer 一种用于序列处理的模型

- 可以消除循环连接
- 全连接网络架构(fully connected networks)

自注

Figure 1: The Transformer - model architecture.

Transformer

映射输入向量序列 $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ 到输出向量序列 $(\mathbf{y}_1, \ldots, \mathbf{y}_n)$

由一堆Transformer块组成，
每一Transformer块是一个多层网络结构：包括

- 简单的linear层，
- 前馈网络，
- 还有self-attention层组合而成

self-attention层

允许网络直接从任意大的上下文中提取和使用信息
不需要通过循环传递信息(如 RNN, LSTM)

self-attention层

核心是可以将感兴趣的输入与之前输入的集合进行比较
计算当前输入在其上下文中的相关性

比较的结果用于计算当前输出

y3是如何计算的？

自注意力机制

y_3计算过程：

计算$x_3 \cdot x_1$、$x_3 \cdot x_2$ 和 $x_3 \cdot x_3$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

计算点积

Softmax归一化处理，获取权重向量

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \ \forall j \leq i \end{aligned}$$

计算输出向量y_3

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

自注意力机制

输入向量在自注意力过程中扮演的三种不同角色

- Query: 当前输入与先前输入进行比较时，查询当前输入重要性程度
- Key: 作为先前输入与当前输入比较时，是参照角色
- Value: 用于计算当前输入所对应的输出的值。

自注意力机制

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i;\ \ \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i;\ \ \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

三个权重矩阵的向量空间大小为 $\mathbb{R}^{d\times d}$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \longrightarrow \text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j \longrightarrow \text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

查询向量点乘键向量

Softmax

$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij}\mathbf{v}_j$$

每一个点积与对应的值向量的点积

当前输出y_3计算流程图

自注意力机制

输出y1…yn，之间的计算是相互独立的

X中的每一行代表一个token的词向量

- 通过将输入序列的N个单词向量打包为一个矩阵 $\mathbf{X} \in \mathbb{R}^{N \times d}$
- 然后利用矩阵计算实现自注意力机制的并行化

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

$\mathbf{Q} \in \mathbb{R}^{N \times d}$ $\mathbf{K} \in \mathbb{R}^{N \times d}$ $\mathbf{V} \in \mathbb{R}^{N \times d}$

查询矩阵与键矩阵计算

$\mathbf{QK}^{\mathsf{T}} =$

N

q1•k1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
q2•k1	q2•k2	$-\infty$	$-\infty$	$-\infty$
q3•k1	q3•k2	q3•k3	$-\infty$	$-\infty$
q4•k1	q4•k2	q4•k3	q4•k4	$-\infty$
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

$(N \times d)$

$$SelfAttention(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{QK}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

比较矩阵的上三角部分归零

Transformer块

残差连接(residual connection)

残差连接是指将信息从较低层，不经过中间层，而直接传递到较高层。

- 来自激活函数的信息向前传递，
- 梯度信息跳过一层向后传递而改善学习。

Transformer 中的残差连接实现是通过

- 将输入向量添加到对应的输出向量，
- 再将添加后的输出向量传递到后续的网络层中。

层归一化(layer normalization)

$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFNN}(\mathbf{z}))$$

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttn}(\mathbf{x}))$$

层归一化(layer normalization)

对所有向量计算均值和标准差

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i-\mu)^2}$$

对向量进行归一化更新

$$\hat{\mathbf{x}} = \frac{(\mathbf{x}-\mu)}{\sigma}$$

引入增益(gain) γ 和偏移值(offset) β 两个可学习参数

$$LayerNorm = \gamma\hat{\mathbf{x}} + \beta$$

多头注意力机制(Multihead Attention)

为什么要用多头注意力？

- 句子单词之间可以以多种方式互相关联，例如
- 不同的句法，语义和语篇关系可以在句子中的动词和相应的论点之间保持。
- 单个transformer块很难学习并捕获这些多重的并行关系。
- Transformer 使用了多个自注意力层来解决(多头 multihead)。
- 不同的注意力层使用不同的参数集合，训练各自的参数矩阵，
- 学习相同输入的不同关系。

多头注意力机制(Multihead Attention)

一个head i 由对应的三个参数矩阵提供

Key query value

$$\mathbf{W}_i^K,\ \mathbf{W}_i^Q \text{ and } \mathbf{W}_i^V$$

$$\mathbb{R}^{d\times d_k} \qquad \mathbb{R}^{d\times d_v}$$

X 是词向量矩阵，空间大小为N x d
其中N为词数量，d为词向量维度

$$MultiHeadAttn(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q\ ;\ \mathbf{K} = \mathbf{X}\mathbf{W}_i^K\ ;\ \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$

$$\mathbf{head}_i = SelfAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

$$\mathbf{Q} \in \mathbb{R}^{N\times d_k} \qquad \mathbf{K} \in \mathbb{R}^{N\times d_k} \qquad \mathbf{V} \in \mathbb{R}^{N\times d_v} \qquad \mathbf{W}^O \in \mathbb{R}^{hd_v\times d}$$

多头注意力机制(Multihead Attention)

词序建模：位置编码(positional encoding)

RNN本身具有顺序特性，
但transformer 没有

如果将输入顺序打乱会怎样？
如：输入顺序为x_1, x_3, x_2

如何解决？

使用位置编码技术生成位置向量，对词向量进行修改更新，使词向量带有位置信息

绝对位置编码生成的位置向量

Transformer 语言模型

下一单词预测

对比：RNN语言模型

Next word
long
and
thanks
for
all
Loss
$-\log y_{\text{long}}$
$-\log y_{\text{and}}$
$-\log y_{\text{thanks}}$
$-\log y_{\text{for}}$
$-\log y_{\text{all}}$
...
$\frac{1}{T}\sum_{t=1}^{T} L_{CE}$
y
Softmax over Vocabulary
Vh
h
RNN
...
Input Embeddings
e
...
So
long
and
thanks
for

Transformer 上下文生成(Contextual Generation)

文本总结(Text summarization)

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!"

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for $19.99, but the snow usually melted before it reached its destination...

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

文本总结(Text summarization)

Transformer的其他使用场景

- 序列的标注任务包括：词性标注(part of speech tagging)或者
  命名实体识别(named entity recognition)

- 序列的分类任务包括：情感分类
  主题分类等

预训练pretraining 和 微调 fine-tune

本章总结

RNN	隐藏层，时间反向传播
LSTMs	遗忘门，输入门，输出门
Transformer	多头自注意力机制