* **RNN以及梯度爆炸：**

图示, 形状

描述已自动生成

图中 **x**、**h**、**y** 分别代表 RNN 神经元的输入、隐藏状态、输出。  
**U**、**W**、**V** 是对向量 **x**、**h**、**y** 进行线性变换的矩阵。  
在 RNN 中每一时刻都共用同一个神经元，将神经元展开之后如下图所示。

图示

描述已自动生成

RNN 中输入 **x**t 只包含 t 时刻信息，不包含顺序信息；而 **h**t 是根据 **x**t 和 **h**t-1 计算得到的，包含了历史信息与当前输入信息。**h**t 与 **y**t 的计算方法如下，计算 **h**t 时激活函数通常采用 tanh，计算输出 **y**t 时激活函数通常是softmax (分类)。

图示

中度可信度描述已自动生成

RNN 的缺陷 (梯度消失与梯度爆炸)

图示

描述已自动生成

我们先看一下只有 3 个输入数据的序列，如上图所示。此时我们的隐藏层 **h**1、**h**2、**h**3 和输出 **y**1、**y**2、**y**3 的计算公式：

文本

描述已自动生成

RNN 在时刻 t 的损失函数为 **L**t，总的损失函数为 **L** = **L**1 + **L**2 + **L**3。

文本

低可信度描述已自动生成

t = 3 时刻的损失函数 **L**3 对于网络参数 **U**、**W**、**V** 的梯度如下：

文本, 信件

描述已自动生成

可以看到对于参数矩阵 **V** (对应输出 **y**t) 的梯度并没有长期依赖，只与 t = 3 时刻的序列相关。但是参数矩阵 **U** (对应输入 **x**t) 和参数矩阵 **W** (对应隐藏状态 **h**t) 的梯度均有长期依赖，依赖于之前的隐藏层状态 **h**1、**h**2。可以推导出时刻 t 的损失函数 **L**t 对于 **U**、**W** 的梯度如下：

文本

描述已自动生成

其中的**连乘项**就是导致 RNN 出现梯度消失与梯度爆炸的罪魁祸首，**连乘项**可以如下变换：

图示

描述已自动生成

tanh' 表示 tanh 的导数，可以看到 RNN 求梯度的时候，实际上用到了 (tanh' × **W**) 的连乘。当 (tanh' × **W**) > 1 时，多次连乘容易导致梯度爆炸；当 (tanh' × **W**) < 1 时，多次连乘容易导致梯度消失。

因为 RNN 计算梯度时候的问题，所以 LSTM (长短期记忆网络) 就出现了。LSTM 在很多方面都碾压了 RNN，并且可以很好地缓解**梯度消失**与**梯度爆炸**的问题。

LSTM与GRU的存在是为了解决简单RNN面临的**长期依赖问题**（由于反向传播存在的梯度消失或爆炸问题，简单RNN很难建模长距离的依赖关系），一种比较有效的方案是在RNN基础上引入**门控机制**来控制信息的传播。

* **LSTM长短期记忆网络**

LSTM通过引入了三个门来控制信息的传递，分别是遗忘门 ，输入门和输出门 。三个门的作用为：

（1）遗忘门  控制**上一时刻的内部状态** 需要遗忘多少信息；

（2）输入门  控制**当前时刻的候选状态**  有多少信息需要保存；

（3）输出门  控制**当前时刻的内部状态** 有多少信息需要输出给**外部状态**  ；

原理图如下所示，从图中理解更加一目了然。

图示

低可信度描述已自动生成

接下来介绍下计算过程和公式，理解LSTM的关键，边看图边理解更快~

**计算过程**

1. 先利用上一时刻的外部状态 和当前时刻的输入 ，计算出三个门的值（式1.1-1.3），以及候选状态 （式1.4）;
2. 结合遗忘门   和输入门  来更新内部状态  ，也称为**记忆单元**；(记住遗忘门  是针对上一时刻的内部状态，输入门 是针对当前时刻的候选状态  ，如式1.5)
3. 结合输出门  ，将内部状态  的信息传递给外部状态  (式1.6)。

**公式**

文本

中度可信度描述已自动生成

以上6个公式就是LSTM的核心了，大家一定要牢记，抓住规律和LSTM的计算过程其实不会太难记的。

文本

描述已自动生成

再回到标题中来，长短期记忆，长短期是什么呢，从上述原理和公式好像并没有体现诶。

其实，传统RNN中的存储着历史信息，但是  每个时刻都会被重写，因此可以看做一种**短期记忆**。**长期记忆**可以看做是网络内部的某些参数，隐含了从数据中学到的**经验**，其更新周期要远远比短期记忆慢。

而在LSTM网络中，内部状态  可以在某个时刻捕捉关键信息，并有能力将此关键信息保存一定的时间间隔，看式1.5，如何保存关键信息可以通过遗忘门  和输入门 进行控制，因此内部状态 保存信息的周期要长于短期记忆，但又要短于长期记忆，因此成为**长短期记忆**，即指**长的“短期记忆”**。

* **GRU 门控循环单元网络**

GRU的结构比LSTM更为简单一些，GRU只有两个门，更新门 和重置门  。

1. 更新门  ：控制**当前状态**  需要从**上一时刻状态** 中保留多少信息（不经过非线性变换），以及需要从**候选状态**中接受多少信息；
2. 重置门 ：用来控制**候选状态**  的计算是否依赖**上一时刻状态**；

文本, 信件

描述已自动生成

图示, 示意图

描述已自动生成

从式2.4分析，GRU直接使用更新门来控制输入和遗忘的平衡，而LSTM中输入门和遗忘门相比GRU就具有一定的冗余性了。可以看出，当 时，当前状态 和上一时刻状态  为非线性关系；当  时，  和 为线性关系。

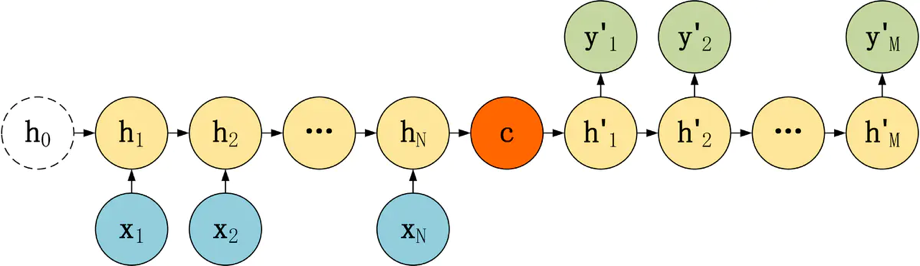
GRU主要记住上述四个公式就基本掌握了其原理。

* **Seq2Seq**

Seq2Seq 是一种循环神经网络的变种，包括编码器 **(Encoder)** 和解码器 **(Decoder)** 两部分。可以用于机器翻译、对话系统、自动文摘。

Seq2Seq 是一种重要的 RNN 模型，也称为 Encoder-Decoder 模型，可以理解为一种 **N×M** 的模型。模型包含两个部分：**Encoder** 用于编码序列的信息，将任意长度的序列信息编码到一个向量 **c** 里。而 **Decoder** 是解码器，解码器得到上下文信息向量 **c** 之后可以将信息解码，并输出为序列。Seq2Seq 模型结构有很多种，下面是几种比较常见的：

**第一种**



第一种 Decoder 结构比较简单，将上下文向量 **c** 当成是 RNN 的初始隐藏状态，输入到 RNN 中，后续只接受上一个神经元的隐藏层状态 **h'** 而不接收其他的输入 **x**。第一种 Decoder 结构的隐藏层及输出的计算公式：

文本

描述已自动生成

**第二种**

卡通人物

中度可信度描述已自动生成

第二种 Decoder 结构有了自己的初始隐藏层状态 ，不再把上下文向量 **c** 当成是 RNN 的初始隐藏状态，而是当成 RNN 每一个神经元的输入。可以看到在 Decoder 的每一个神经元都拥有相同的输入 **c**，这种 Decoder 的隐藏层及输出计算公式：

文本

描述已自动生成

**第三种**

图示

描述已自动生成

第三种 Decoder 结构和第二种类似，但是在输入的部分多了上一个神经元的输出 **y'**。即每一个神经元的输入包括：上一个神经元的隐藏层向量 **h'**，上一个神经元的输出 **y'**，当前的输入 **c** (Encoder 编码的上下文向量)。对于第一个神经元的输入 ，通常是句子其实标志位的 embedding 向量。第三种 Decoder 的隐藏层及输出计算公式：

文本

描述已自动生成

这三种 Seq2Seq 模型的主要区别在于 Decoder，他们的 Encoder 都是一样的。下图是 Encoder 部分，Encoder 的 RNN 接受输入 **x**，最终输出一个编码所有信息的上下文向量 **c**，中间的神经元没有输出。Decoder 主要传入的是上下文向量 **c**，然后解码出需要的信息。

卡通画

中度可信度描述已自动生成

从上图可以看到，Encoder 与一般的 RNN 区别不大，只是中间神经元没有输出。其中的上下文向量 **c** 可以采用多种方式进行计算。

文本

描述已自动生成

从公式可以看到，**c** 可以直接使用最后一个神经元的隐藏状态 表示；也可以在最后一个神经元的隐藏状态上进行某种变换 而得到，q 函数表示某种变换；也可以使用所有神经元的隐藏状态 **h**1, **h**2, ..., 计算得到。得到上下文向量 **c** 之后，需要传递到 Decoder。

**Seq2Seq模型使用技巧**

* **Teacher Forcing**

Teacher Forcing 用于训练阶段，主要针对上面第三种 Decoder 模型来说的，第三种 Decoder 模型神经元的输入包括了上一个神经元的输出 **y'**。如果上一个神经元的输出是错误的，则下一个神经元的输出也很容易错误，导致错误会一直传递下去。

而 Teacher Forcing 可以在一定程度上缓解上面的问题，在训练 Seq2Seq 模型时，Decoder 的每一个神经元并非一定使用上一个神经元的输出，而是有一定的比例采用正确的序列作为输入。

举例说明，在翻译任务中，给定英文句子翻译为中文。"I have a cat" 翻译成 "我有一只猫"，**下图是不使用 Teacher Forcing 的 Seq2Seq**：

图示

描述已自动生成

**如果使用 Teacher Forcing**，则神经元直接使用正确的输出作为当前神经元的输入。

图示

描述已自动生成

* **Attention**

在 Seq2Seq 模型，Encoder 总是将源句子的所有信息编码到一个固定长度的上下文向量 **c** 中，然后在 Decoder 解码的过程中向量 **c** 都是不变的。这存在着不少缺陷：

* 对于比较长的句子，很难用一个定长的向量 **c** 完全表示其意义。
* RNN 存在长序列梯度消失的问题，只使用最后一个神经元得到的向量 **c** 效果不理想。
* 与人类的注意力方式不同，即人类在阅读文章的时候，会把注意力放在当前的句子上。

Attention 即**注意力机制**，是一种将模型的注意力放在当前翻译单词上的一种机制。例如翻译 "I have a cat"，翻译到 "我" 时，要将注意力放在源句子的 "I" 上，翻译到 "猫" 时要将注意力放在源句子的 "cat" 上。

使用了 Attention 后，Decoder 的输入就不是固定的上下文向量 **c** 了，而是会根据当前翻译的信息，计算当前的 **c**。

图示, 示意图

描述已自动生成

Attention 需要保留 Encoder 每一个神经元的隐藏层向量 **h**，然后 Decoder 的第 t 个神经元要根据上一个神经元的隐藏层向量 计算出当前状态与 Encoder 每一个神经元的相关性 **e**t。**e**t 是一个 N 维的向量 (Encoder 神经元个数为 N)，若 **e**t 的第 i 维越大，则说明当前节点与 Encoder 第 i 个神经元的相关性越大。**e**t 的计算方法有很多种，即相关性系数的计算函数 a 有很多种：

文本, 信件

描述已自动生成

上面得到相关性向量 **e**t 后，需要进行归一化，使用 softmax 归一化。然后用归一化后的系数融合 Encoder 的多个隐藏层向量得到 Decoder 当前神经元的上下文向量 **c**t：

文本, 信件

描述已自动生成

* **beam search（集束搜索）**

beam search 方法不用于训练的过程，而是用在测试的。在每一个神经元中，我们都选取当前输出概率值最大的 **top k** 个输出传递到下一个神经元。下一个神经元分别用这 k 个输出，计算出 L 个单词的概率 (L 为词汇表大小)，然后在 kL 个结果中得到 **top k** 个最大的输出，重复这一步骤。

* **Transformer**

Transformer最开始提出是用来解决机器翻译任务，因此可以看作是seq2seq模型的一种。从上节的seq2seq框架知识我们可以发现，Transformer宏观结构属于seq2seq范畴，只是将之前seq2seq中的编码器和解码器，从RNN模型替换成了Transformer模型。

图示

描述已自动生成

图中左边部分为Encoder 部分，右边部分为Decoder部分和最后的线性输出层。其中Encoder和Decoder各有6层。Transformer的输入：word\_embedding + positional\_embedding。

* **Encoder**

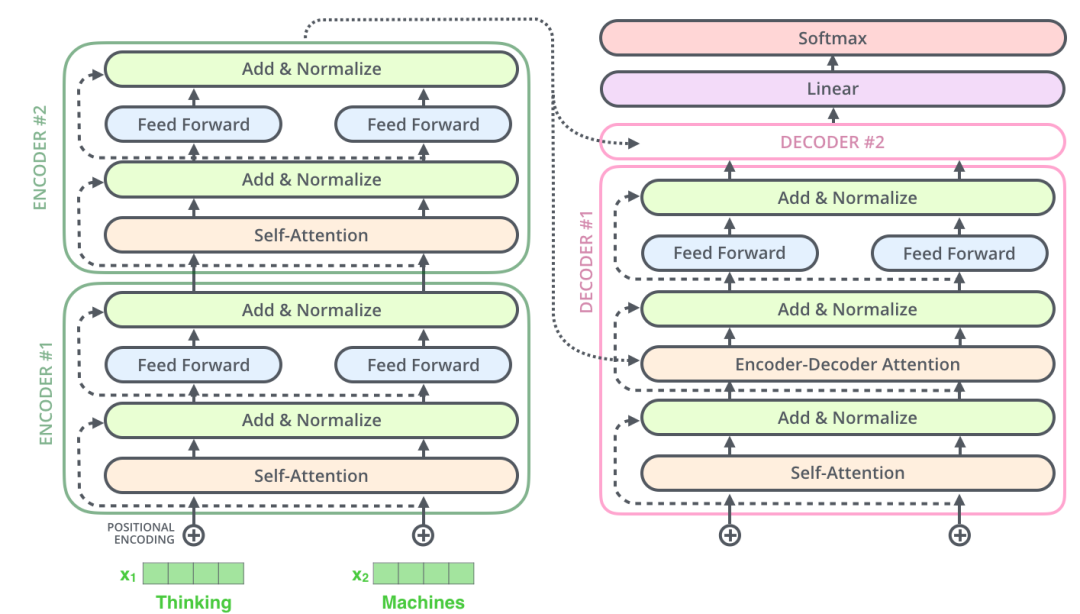
对于Encoder部分来说，每一层里面有两层。分别是一层self-attention层和一层全连接层。需要注意的是，这里的self-attention并不是只有一层。模型中使用的是multi-head-Attention。其实就是多个self-attention，可以把每个self-attention理解为一个head。把每一个self-attention的输出结果拼接起来。然后输入给后面的全连接网络。全连接网络层是一个两层的网络，第一层是Relu激活函数，第二层是一个线性的激活函数。前面几层的encoder的输出，会作为输入给下一层的encoder。每一个encoder里的两层的输出，都会进入一个add&Norm。最后的encoder会输出给后面的decoder模型。

* **Decoder**

Decoder部分和Encoder一样，也是有6层，但是每一个单独的decoder与encoder相比，在self-attention层（decoder层中叫masked self-attention）和全连接网络层之间，多了一层Encoder-Decoder-Attention 层。decoder结构中，第一层是一个multi-head-self-attention层，这个与encoder中的区别是这里是masked-multi-head-self-attention。使用mask的原因是因为在预测句子的时候，当前时刻是无法获取到未来时刻的信息的。

decoder中的第二层attention层就是一个正常的multi-head attention层。但是这里Q,K,V来源不同。Q来自于上一个decoder的输出，而K,V则来自于encoder的输出。

关于这两个attention层，可以理解为 mask-self-attention是计算当前翻译的内容和已经翻译的前文之间的关系，而encoder-decoder-attention 是计算当前翻译内容和编码的特征向量之间的关系。最后再经过一个全连接层，输出decoder的结果。



* **残差连接**

到目前为止，我们计算得到了self-attention的输出向量。而单层encoder里后续还有两个重要的操作：残差链接、标准化。编码器的每个子层（Self Attention 层和 FFNN）都有一个残差连接和层标准化（layer-normalization），如上图所示。

为什么需要shortcut connection？

1. 优化梯度回传

在梯度更新的步骤，我们一般使用反向传播法（backpropagation）来计算损失函数对于权重的梯度，并加以更新。然而在深层的神经网络中，由于各种原因（比如激活函数的饱和性（saturated）、权重的分布不佳等等），梯度在反向传播的过程中，信号会逐渐消失，这会导致顶层权重更新速度快，而底层权重几乎不更新的情况。这是灾难性的，因为底层所提取到的特征往往更为重要，例如边缘、纹理等等更为普遍的性质，从而使得训练效果不佳。

除此之外，在一般的神经网络中，如果某一层停止学习，则信号就会在此消失，而无法传到更底层，从而造成“梯度消失”现象。

然而，在shortcut connection的存在下，梯度回传时，即使几层还没有开始学习（残差函数的梯度接近零），由于恒等函数的导数恒为1，整个函数的梯度依然接近于1，根据链式法则，先前的梯度依然可以反向传播，网络也可以开始取得进展（见下图）。借助shortcut connection，信号可以轻松地在整个网络中传播，从而避免了“梯度消失”。

图示, 工程绘图

描述已自动生成

1. 残差函数训练更容易

让我们复习一下：在训练神经网络时，目标函数为H(x)的模型。如果将输入x（即恒等函数）添加到网络的输出，则网络被迫建模残差函数F(x)，这称为**残差学习**。

如前所述，ResNet基于这样一种假设：最优函数与线性函数有一定的相似性。初始化常规神经网络时，其权重参数接近零，因此网络仅输出其输入的副本。换言之，它首先对恒等函数建模。所以如果目标函数和恒等函数相当接近（通常是这种情况），那么训练速度会大大加快。这也是残差学习相对更加容易的原因。

此外，利用残差块可以训练出一个有效的深层神经网络：输入可以通过层间的残余连接更快地向前传播。

1. 嵌套函数类的实现

ResNet的提出，最初是为了解决“负优化”问题，也就是说让更深层的网络表现的至少和浅层网络一样好。这就涉及函数类的概念。

假设我们有一类特定的神经网络架构 F ，它包括学习率及其他超参数。对于所有，存在一些参数集（包括权重和偏置），这些参数可以通过在合适的训练集上进行训练所获得。现在假设 是我们真正想要找到的函数，如果，那么我们可以轻而易举地获得它，但通常我们不会那么幸运。相反，我们将尝试找到一个函数 ，这是我们在 F 中的最佳选择。例如，给定一个具有 X 特性和 y 标签的数据集，我们可以尝试通过解决以下优化问题来找到它：。  
那么，怎样得到更近似真正 的函数呢？唯一合理的可能性是，我们需要设计一个更强大的架构。换句话说，我们预计  比  “更近似”。然而，如果，则无法保证新的体系“更近似”。事实上，  可能更糟：如图所示，对于非嵌套函数(non-nested function)类，较复杂的函数类并不总是向“真”函数靠拢(复杂度由 F1 向 F6 递增)。在下图的左边，虽然 F3 比 F1 更接近 ，但 F6 却离得更远了。相反对于下图右侧的嵌套函数(nested function)类 F1∈⋯∈F6 ，我们可以避免上述问题。

卡通人物

中度可信度描述已自动生成

因此，只有当较复杂的函数类包含较小的函数类时，我们才能确保提高它们的性能。对于深度神经网络，如果我们能将新添加的层训练成恒等映射，新模型将和原模型同样有效。同时，由于新模型可能得出更优的解来拟合训练数据集，因此添加层似乎更容易降低训练误差。

如上所述，通过加入shortcut connection，我们构造出了一系列“嵌套函数类”，使得层数越多，训练效果越好。

* **Layer Normalization**

图示

描述已自动生成

BN（batch normalization）本质上来说是一个归一化层，归一化层在神经网络中应用广泛，为什么需要有归一化层呢？首先在训练前，每个batch的数据分布可能有很大差别，如果模型按照每个模型的真实数据分布进行学习和迭代，那么可能会大大降低收敛速度。而且训练数据和测试数据的分布也可能不同，所以数据在训练前经过归一化预处理可以提高模型的**训练速度**和**泛化能力**。其次在训练过程中，数据经过神经网络前几层可能会发现一些细微的变化，而这个变化经过后面多层的传播，可能会越来越发散，训练数据的分布一直在变化的话，模型的每一层都要重新学习和迭代，这样也会影响到模型训练速度，这个现象就叫漂移。

BN层的作用就是在神经网络中的每层之前插入归一化层，对输入的数据进行归一化处理，对每一个batch都进行操作。例如现在有一个batch的数据，包含了两个特征（身高、体重），共10条数据。那么BN会对这个batch的身高特征进行缩放，算出身高的均值和方差，然后对10个身高数据进行缩放，体重同理，相当于对batch的每列数据分别进行缩放。

如果我们将一批文本组成一个batch，那么BN的操作方向是，对每句话的**第一个**词进行操作。但语言文本的复杂性是很高的，任何一个词都有可能放在初始位置，且词序可能并不影响我们对句子的理解。而BN是**针对每个位置**进行缩放，这**不符合NLP的规律**。

而LN则是针对一句话进行缩放的，且L**N一般用在第三维度**，如[batchsize, seq\_len, dims]中的dims，一般为词向量的维度，这一维度各个特征的量纲应该相同。因此也不会遇到上面因为特征的量纲不同而导致的缩放问题。BN用在NLP中并不是对每句话的第一个字进行归一化，而是对每句话的所有字向量上第一个维度的数做归一化。

* **Positional Encoding**

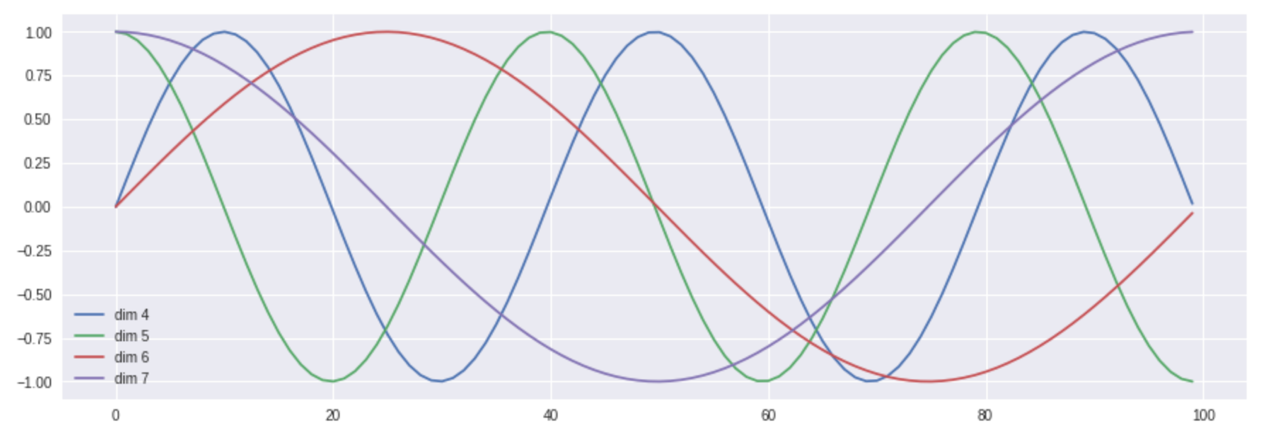
因为按照attention的计算公式，我们只不过是计算了两两之间的attention值。‘I have a dream’ 和 ‘Dream have a I’这两句话对于attention而言是一样。那么截止目前为止，模型还不过是一个复杂的词袋模型，没有考虑到词序。而RNN这种模型的一大特点就是考虑到了词序。为了解决这个问题，transformer模型在数据预处理时就提出了位置编码这个概念。  
原文中对于这个位置编码提出了两种方式，第一种是训练出一个位置编码，第二种是原文中使用的用三角函数编码的方法。具体公式如下

图示

描述已自动生成

在下图中，我们画出了一种位置向量在第4、5、6、7维度、不同位置的的数值大小。横坐标表示位置下标，纵坐标表示数值大小。

图：位置编码在0-100位置，在4、5、6、7维的数值图示



* **Generator部分**

最后生成的部分相比之下简单很多。由一个线性层再加一个softmax层完成最后的输出。

图示

描述已自动生成