

1.

The figure consists of four parts labeled (a) through (d). Part (a) shows a large triangular graph representing a recursive convolutional neural network, where nodes are arranged in layers and connected by edges. Part (b) illustrates the proposed gated unit, showing a node receiving inputs from multiple sources and producing an output. Part (c) shows two example structures that may be learned with the proposed gated unit, each consisting of a small tree-like graph. Part (d) shows another example structure, similar to (c), but with different connections.

门控单元的工作原理:

激活函数:

- 新的激活值 $\tilde{h}_j^{(t)}$ 通过以下方式计算:

$$\tilde{h}_j^{(t)} = \phi(W^l h_{j-1}^{(t)} + W^r h_j^{(t)})$$

- 其中 ϕ 是按元素运算的非线性函数。

门控系数:

- 门控系数 ω 的计算方式如下:

$$\begin{bmatrix} \omega_c \\ \omega_l \\ \omega_r \end{bmatrix} = \frac{1}{Z} \exp(G^l h_{j-1}^{(t)} + G^r h_j^{(t)})$$

- Z 是归一化因子, 确保 ω 的和为1。

网络结构的适应性:

- 网络可以在递归层级 t 根据左右子节点的新激活值、左子节点的激活值或右子节点的激活值中选择其一, 从而使网络结构适应输入样本。
- 这种机制使得网络结构可以适应输入样本, 类似于无监督解析。
- 如果门控单元作出硬决策 (例如使用1-of-K编码), 网络将适应输入并形成树状结构。

图解说明:

文档最后提供了图2, 说明了:

- (a) 递归卷积神经网络的结构。
- (b) 提出的门

AI

Let $x = (x_1, x_2, \dots, x_T)$ be an input sequence, where x_t is a binary vector of dimension d , and T is the length of the sequence. In addition to a usual convolutional architecture, we propose to use the recursively defined gating mechanism which takes the representation $h_j^{(l)}$ and the representation $h_{j+1}^{(l)}$ as inputs.

Let W^l, W^r, G^l and G^r be the weights of the recursive gated unit (grConv) at layer l . At each recursive level $l \in [1, T-1]$, the activation of the j -th hidden unit $h_j^{(l)}$ is computed by

$$h_j^{(l)} = \omega_{c,j}^{(l)} h_j^{(l-1)} + \omega_{l,j}^{(l)} h_{j-1}^{(l-1)} + \omega_{r,j}^{(l)} h_{j+1}^{(l-1)}, \quad (1)$$

$$Z = \frac{1}{2} \left[\exp(G^l h_{j-1}^{(l-1)} + G^r h_{j+1}^{(l-1)}) + 1 \right],$$

where $Z \in [0, 1]$, $0^{T+1} = 0$ and

$$Z = \frac{1}{2} \left[\exp(G^l h_{j-1}^{(l-1)} + G^r h_{j+1}^{(l-1)}) + 1 \right].$$

Since $Z \in [0, 1]$, the activation of the recursive gated unit is a convex combination of the activation from the left and right children. The activation from the left child or the activation from the right child. This is chosen since the overall activation of the recursive unit can be changed adaptively with respect to an input sample. See Fig. 2(a) for an illustration.

The recursive gated unit is illustrated in Fig. 2(b).

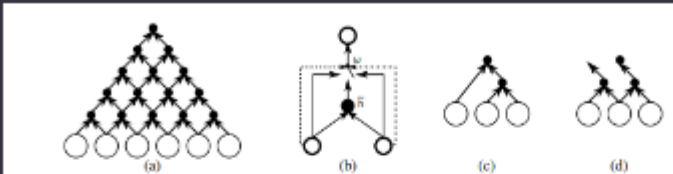


Figure 2: The graphical illustration of (a) the recursive convolutional neural network and (b) the proposed gated unit for the recursive convolutional neural network. (c-d) The example structures that may be learned with the proposed gated unit.

翻译这一段，然后教我，最好举例说明，图解说明



翻译：

2.2 门控递归卷积网络

除了RNNs之外，处理可变长度序列的另一个自然方法是使用一个递归卷积神经网络，其中每一层的参数通过整个网络共享（见图2 (a)）。在本节中，我们介绍了一种二进制递归卷积神经网络，其权重通过输入序列递归应用，直到输出单一固定长度向量。除了通常的卷积架构外，我们提议使用先前提到的门控机制，该机制允许递归网络在运行中学习源句子的结构。

设 $x = (x_1, x_2, \dots, x_T)$ 是输入序列，其中 $x_t \in \mathbb{R}^d$ 。所提出的门控递归卷积网络(grConv)包括四个权重矩阵 W^l, W^r, G^l 和 G^r 。在递归层次 $l \in [1, T-1]$ 上，第 j 个隐藏单元 $h_j^{(l)}$ 的激活是通过以下计算得出的：

$$h_j^{(l)} = \omega_{c,j}^{(l)} h_j^{(l-1)} + \omega_{l,j}^{(l)} h_{j-1}^{(l-1)} + \omega_{r,j}^{(l)} h_{j+1}^{(l-1)}, \quad (1)$$

其中 $\omega_{c,j}, \omega_{l,j}$ 和 $\omega_{r,j}$ 是总和为1的门控单元的值。隐藏单元是这样初始化的：

$$h_j^{(0)} = Ux_j,$$

其中 U 将输入投影到一个隐藏空间。

新的激活 $\tilde{h}_j^{(l)}$ 按照通常的方式计算出：

$$\tilde{h}_j^{(l)} = \phi(W^l h_{j-1}^{(l-1)} + W^r h_{j+1}^{(l-1)}),$$

其中 ϕ 是逐元素非线性函数。门控系数 ω 's 通过以下计算得出：

$$\begin{pmatrix} \omega_{c,j}^{(l)} \\ \omega_{l,j}^{(l)} \\ \omega_{r,j}^{(l)} \end{pmatrix} = \frac{1}{2} \exp(G^l h_{j-1}^{(l-1)} + G^r h_{j+1}^{(l-1)}),$$

Regenerate

$$\begin{pmatrix} \omega_{c,j}^{(l)} \\ \omega_{l,j}^{(l)} \\ \omega_{r,j}^{(l)} \end{pmatrix} = \frac{1}{Z} \exp(G^l h_{j-1}^{(l)} + G^r h_j^{(l)}),$$

其中 $G^l, G^r \in \mathbb{R}^{3 \times d}$ 并且

$$Z = \sum_{k=1}^3 (\exp(G^l h_{j-1}^{(l)} + G^r h_j^{(l)}))_k.$$

根据这种激活，可以将一个节点在递归层次 l 的激活视为选择新的激活组成部分，该部分既可以从左子节点也可以从右子节点获取，亦或是维持来自左子节点的激活或者从右子节点获取激活。这种选择允许递归卷积的整体结构随输入样本适应性变化。参见图2(b)进行说明。

从这个角度看，我们甚至可以将提出的grConv视为一种非监督的解析。如果我们考虑门控单元做出硬性决定的情况，即遵循1-of-K编码，我们很容易看出网络如何做出决策，并形成树状结构以适应输入。然而，我们需要进一步调查这个模型学到的结构的性质。

图2: (a) 递归卷积神经网络的图解；(b) 递归卷积神经网络的门控单元示意图；(c)-(d) 可能通过提出的门控单元学习到的结构示例。

解释与例子：

这段文字介绍了门控递归卷积网络(grConv)，这是一种处理可变长度序列数据的神经网络结构。在这种结构中，网络的权重在递归地应用到输入序列上，从而产生一个固定长度的输出向量。这种网络包括一个门控机制，使网络能够在处理数据时学习数据的内部结构。

例如，如果你有一系列的句子，每个句子由不同数量的单词组成，grConv可以通过其递归结构学习这些句子的语法或语义结构。门控单元控制信息是如何在网络中流动的，哪些信息被传递到下一个层级，哪些被保留或忽略。

在图2中，你可以看到：

- (a) 显示了递归卷积神经网络的整体结构。
- (b) 展示了网络中单个门控单元的操作，如何通过控制左、中、右三个权重的分配来影响信息流。
- (c) 和 (d) 显示了网络可能学习的不同结构的例子，这些结构是根据输入序列的不同特征自动形成的。

例如，如果网络正在处理自然语言句子，它可能会学习到类似于语法树的结构，其中每个节点代表一个单词，它的子节点代表该单词的语法功能，如主语、动词或宾语。通过这种方式，网络不仅学习如何处理序列，而且还学习序列内部的结构。

AI

connections at each level are shared through the whole network (see Fig. 2). (c) In this version, we introduce a binary, non-directional spatial connectivity, where weights are symmetrically applied to the input sequence and it respects a single fixed length rule.

In addition to a usual convolutional operation, we propose to use the recursively defined gating mechanism which allows the network to work at both the bottom of the coarse structure and the top.

Let us fix $\{x_i, y_i, v_i, u_i\}_{i=1}^L$ as input sequence, where $v_i, u_i \in \mathbb{R}^d$. The proposed gated recursive convolutional network consists of two main parts: the two weight matrices $\{W^l, W^r, G^l, G^r\}$ and $\{Q^l, Q^r\}$ and $\{u_c, u_l, u_r\}$ for each convolution and $\{g, \tau\}_{i=1}^L$ for activation of the (de)convolutional layer respectively:

$$Q^{l,r} = u_c^T Q + u_l^T Q + u_r^T Q + u_c^T Q^{l,r}, \quad (1)$$

where Q is used to generate the activation's probability.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \frac{1}{Z} \exp \left\{ G^l v_{i+1}^T + G^r v_i^T \right\}$$

where $G^l, G^r \in \mathbb{R}^{d \times d}$ and

$$Z = \sum_{i=1}^L \left[\exp \left\{ G^l v_{i+1}^T + G^r v_i^T \right\} \right].$$

According to the activation, one can think of the activation of a single node at recursive level i as a choice between either a pair activation computed from both left and right children, the activation from the left child, or the activation from the right child. This choice allows the event if a node of the recursive network is change adaptively with respect to an input sample. See Fig. 3 for an illustration.

In this paper, we propose to consider the pro-

你好，还有这里说的四个权重矩阵，我没理解它都代表什么权重矩阵，重置门、更新门的还是什么的？



这里描述的是一种门控卷积神经网络（Gated Convolutional Neural Network），它结合了循环神经网络（RNN）的门控机制（例如，在LSTM中的忘记门和输入门）与卷积神经网络（CNN）的空间特征提取能力。这种结构特别适合处理具有变长序列的数据。



文中提到的四个权重矩阵 W^l, W^r, G^l 和 G^r 具有不同的作用：

- W^l 和 W^r : 这两个权重矩阵分别用于在递归卷积操作中处理左子节点和右子节点的信息。在此上下文中，左子节点和右子节点可能是指在序列或树状结构中相邻的元素或节点，而 l 和 r 分别指左（left）和右（right）。
- G^l 和 G^r : 这两个权重矩阵用于计算门控系数，即 u_c, u_l 和 u_r ，它们类似于LSTM中的门控机制。门控系数决定了节点应该保留多少来自左子节点和右子节点的信息（或者是以前状态的信息），以及新激活的信息应该有多少被加入到当前节点的状态中。

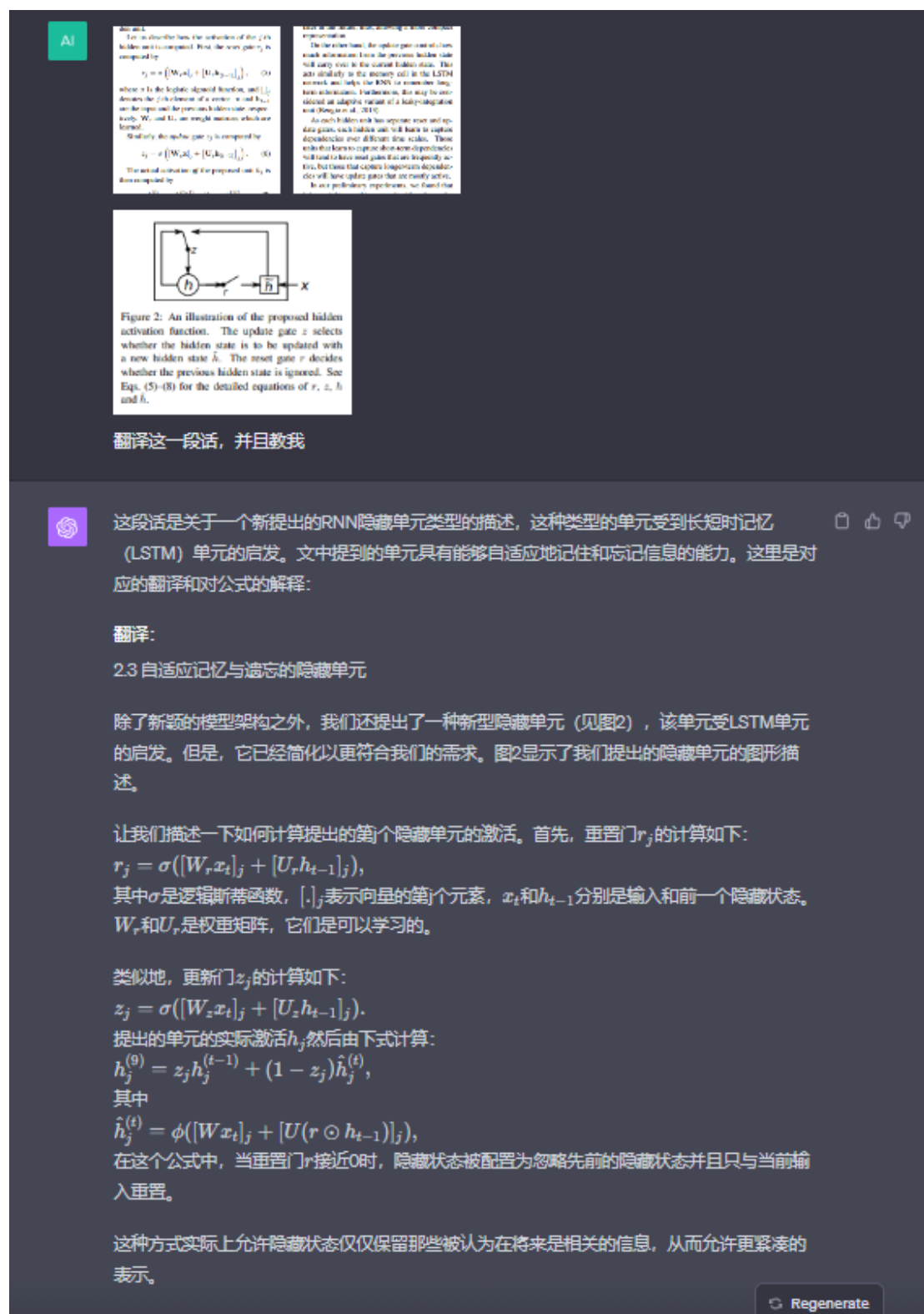
这些门控系数的计算方式如下：

$$\begin{aligned} u_c &= \frac{1}{Z} \exp(G^l h_{j+1}^{(t)} + G^r h_j^{(t)}) \\ u_l &= \frac{1}{Z} \exp(G^l h_{j-1}^{(t-1)}) \\ u_r &= \frac{1}{Z} \exp(G^r h_j^{(t-1)}) \end{aligned}$$

这里 $h_j^{(t)}$ 表示当前节点在时间 t 的激活，而 $h_{j-1}^{(t-1)}$ 和 $h_j^{(t-1)}$ 分别表示左子节点和右子节点在时间 $t-1$ 的激活。 Z 是一个归一化因子，确保 u_c, u_l 和 u_r 的和为1。

总之， W 系列矩阵负责处理节点信息，而 G 系列矩阵负责调节信息流动，这有点类似于LSTM中的忘记门和更新门的作用。通过这种机制，网络可以在处理序列数据时动态调整其结构。

2.



这种方式实际上允许隐藏状态仅仅保留那些被认为在将来是相关的信息，从而允许更紧凑的表示。

另一方面，更新门控制了多少信息从先前的隐藏状态应该传递到当前的隐藏状态。这有助于RNN记住长期信息，并且让记忆细胞在LSTM网络中起到类似的作用。这可能被视为渗漏积分单元的一种自适应形式。

由于每个隐藏单元都有各自的重置和更新门，每个单元将学习在不同时间尺度上捕获依赖性。那些经常被激活的重置门将倾向于捕获短期依赖性，而那些更新门活跃的将倾向于捕获长期依赖性。在我们的初步实验中，我们发现在没有任何门控制的情况下，我们无法获得有意义的结果。

图解：

图2是所提出的隐藏单元激活函数的示意图。更新门 z 选择是否要更新隐藏状态以及用一个新的隐藏状态 \hat{h} 。重置门 r 决定是否忽视先前的隐藏状态。更详细的 r 、 z 、 h 和 \hat{h} 的计算请参见公式(5)至(8)。

教学：

在这个模型中，每个时间步的每个隐藏单元的状态更新依赖于两个门控机制：

1. **重置门 r** ：这决定了多少以前的隐藏状态将被考虑在内。当 r 接近0时，几乎会忽略之前的隐藏状态。
2. **更新门 z** ：这控制了隐藏状态的更新幅度。当 z 接近1时，几乎只保留旧的状态；当 z 接近0时，新的信息 \hat{h} 将完全影响隐藏状态。
3. **实际激活 h** ：新的隐藏状态是通过更新门 z 和重置门 r 以及当前的输入和先前的隐藏状态来计算得出的。

这样的门控机制使得模型能够更好地学习和利用长期和短期的依赖关系，这对于处理序列数据如自然语言处理或时间序列分析来说是非常重要的。

AI




Figure 1. The architecture of a standard recurrent neural network (RNN) for processing a sequence of inputs x_1, x_2, \dots, x_T . At each time step t , the input x_t is combined with the hidden state h_{t-1} from the previous step to produce the hidden state h_t . The hidden state is then used to produce the output y_t . The hidden state is updated according to the equation $h_t = f(h_{t-1}, x_t)$, where f is a function that takes the previous hidden state and the current input as inputs and produces the next hidden state as output. The output y_t is produced by a function g that takes the hidden state h_t as input and produces the output y_t as output. The hidden state is initialized to a zero vector $h_0 = \mathbf{0}$. The output y_t is used to calculate the loss \mathcal{L} for the sequence. The loss is then used to calculate the gradients of the loss with respect to the weights and biases of the network, which are used to update the weights and biases using a gradient descent algorithm.








Figure 2. The architecture of a Gated Recurrent Unit (GRU) for processing a sequence of inputs x_1, x_2, \dots, x_T . The GRU consists of two gates: a reset gate r_t and an update gate z_t . The reset gate r_t is used to determine how much of the previous hidden state h_{t-1} to forget. The update gate z_t is used to determine how much of the previous hidden state h_{t-1} to update. The reset gate r_t and the update gate z_t are both produced by a function σ that takes the previous hidden state h_{t-1} and the current input x_t as inputs and produces the reset gate r_t and the update gate z_t as outputs. The reset gate r_t is used to calculate the candidate hidden state \tilde{h}_t according to the equation $\tilde{h}_t = \tanh(W_h \tilde{h}_{t-1} + W_x x_t)$, where $\tilde{h}_{t-1} = (1 - r_t) \odot h_{t-1} + r_t \odot \tilde{h}_{t-1}$ and \odot is the element-wise product. The update gate z_t is used to calculate the final hidden state h_t according to the equation $h_t = z_t \odot \tilde{h}_t + (1 - z_t) \odot h_{t-1}$. The hidden state h_t is then used to produce the output y_t by a function g that takes h_t as input and produces y_t as output. The hidden state is initialized to a zero vector $h_0 = \mathbf{0}$. The output y_t is used to calculate the loss \mathcal{L} for the sequence. The loss is then used to calculate the gradients of the loss with respect to the weights and biases of the network, which are used to update the weights and biases using a gradient descent algorithm.

可以告诉我这段内容详细



这段内容是关于可处理可变长度序列的神经网络的描述，特别是关于递归神经网络（Recurrent Neural Networks, RNNs）的。这里提到了RNN及其变种，它们可以通过维持隐藏状态来处理序列数据。



以下是该段内容的详细解释：

2.1 节点介绍

2.1.1 递归神经网络（Recurrent Neural Network）

递归神经网络（RNN）如图1（a）所示，能够处理变长序列 $x = (x_1, x_2, \dots, x_T)$ 通过在时间上维持一个隐藏状态 h 。在每个时间步 t ，隐藏状态 $h^{(t)}$ 通过以下公式更新：

$$h^{(t)} = f(h^{(t-1)}, x_t)$$

这里 f 是一个激活函数，通常是通过对输入向量进行线性变换，将它们求和，然后应用一个元素级的逻辑斯蒂激活函数。

RNN能够通过学习序列中下一个输入的分布 $p(x_{t+1}|x_t, \dots, x_1)$ 来有效地学习可变长度序列上的分布。举例来说，在一个序列是1-of-K向量的情况下，RNN的输出可以学习以下分布：

$$p(x_{t,j} = 1|x_{t-1}, \dots, x_1) = \frac{\exp(w_j h^{(t)})}{\sum_{j=1}^K \exp(w_j h^{(t)})}$$

其中 $j = 1, \dots, K$ ， w_j 是权重矩阵 W 的行。这会产生整个序列上的联合分布：

$$p(x) = \prod_{t=1}^T p(x_t|x_{t-1}, \dots, x_1)$$

2.1.2 带门控隐藏神经元的RNN

最近，Cho等人在2014年提出了一种新的RNN激活函数。这个新的激活函数在常规的逻辑斯蒂激活函数上增加了两个门控单元：重置门 r 和更新门 z 。每个门控单元都依赖于前一个隐藏状态 $h^{(t-1)}$ ，而当前输入 x_t 控制信息流。这与长短期记忆单元（LSTM units）有相似之处。为了更详细的信息，作者指引读者查看Cho等人在2014年的工作和图1（b）。

从这段内容可以看出，作者在讨论RNN如何适用于序列预测问题，并介绍了一种新的RNN变体，该变体使用了类似LSTM的门控机制来更有效地处理长期依赖关系。

图1(a)展示的是标准RNN的结构，其中每个箭头代表了时间步中的数据流动。图1(b)则提供了含门控单元的隐藏层的一个抽象视图，说明了隐藏状态是如何通过结合当前输入和前一个隐藏状态来更新的。门控单元允许模型决定何时更新隐藏状态（通过更新门）以及何时重置隐藏状态（通过重置门），从而在处理序列数据时更有效地记住和遗忘信息。

🔄 Regenerate

3. RNN Encoder Decoder

RNN Encoder-Decoder

In this paper, we propose a novel neural network architecture that learns to encode a variable-length input into a fixed-length vector representation and to decode a given fixed-length vector representation back into a variable-length sequence. From a probabilistic perspective, this new architecture can be viewed as a general method to learn the conditional distribution over a variable-length sequence y given a fixed-length sequence x , i.e., $p(y_1, \dots, y_T | x_1, \dots, x_T)$, where θ is the set of the model parameters and each (x_n, y_n) is an (input sequence, output sequence) pair from the training set. In our case, as the output of the decoder, starting from the input, is differentiable, we can use a gradient-based algorithm to estimate the model parameters.

Once the RNN Encoder-Decoder is trained, the model can be used in two ways. One way is to use the model to generate a target sequence given an input sequence. On the other hand, the model can be used to score a given pair of input and output sequences, where the score is simply a probability $p(y | x)$ from Eqs. (3) and (4).

The encoder is an RNN that reads each symbol of an input sequence x sequentially. As it reads each symbol, the hidden state of the RNN changes according to Eq. (1). After reading the end of the sequence (marked by an end-of-sequence symbol), the hidden state of the RNN is a summary c of the whole input sequence.

The decoder of the proposed model is another RNN which is trained to generate the output sequence by predicting the next symbol y_t given the hidden state h_{t-1} . However, unlike the RNN described in Sec. 2.1, both y_t and h_{t-1} are also conditioned on $y_{1:t-1}$ and on the summary c of the input sequence. Hence, the hidden state of the decoder at time t is computed by

$$h_t = f(h_{t-1}, y_{t-1}, c, y_1, \dots, y_{t-1}),$$

where θ is the set of the model parameters and each (x_n, y_n) is an (input sequence, output sequence) pair from the training set. In our case, as the output of the decoder, starting from the input, is differentiable, we can use a gradient-based algorithm to estimate the model parameters.

Once the RNN Encoder-Decoder is trained, the model can be used in two ways. One way is to use the model to generate a target sequence given an input sequence. On the other hand, the model can be used to score a given pair of input and output sequences, where the score is simply a probability $p(y | x)$ from Eqs. (3) and (4).

翻译这段话，并且教我

这是对循环神经网络 (RNN) 编码器-解码器模型 (Encoder-Decoder Model) 的描述。以下是对应的翻译和解释：

翻译：

2.2 RNN编码器-解码器

在这篇论文中，我们提出了一种新的神经网络架构，它学会将一个可变长度的序列编码成一个固定长度的向量表示，并将这个固定长度的向量表示解码回一个可变长度的序列。从概率论的角度来看，这个新模型是一个通用方法，用来学习一个可变长度序列的条件分布，这个条件是基于一另一个可变长度序列，例如 $p(y_1, \dots, y_T | x_1, \dots, x_T)$ ，其中一个

图1: 所提出的RNN编码器-解码器的插图。

应该注意的是输入和输出序列的长度 T 和 T' 可能不同。编码器是一个RNN，它依次读取每一个符号来编码输入序列 x 。它读取每个符号时，隐藏状态的RNN按照方程 (1) 改变。读取序列末尾 (由一个序列结束符号标记) 之后，RNN的隐藏状态是整个输入序列的总结 c 。

所提出模型的解码器是另一个RNN，它被训练来生成输出序列，预测下一个符号，给定隐藏状态 h_{t-1} 。然而，与之前在2.1节中描述的RNN解码器不同， y_t 和 h_t 同时也是由 h_{t-1} 和序列输入的总结 c 决定的。因此，解码器的隐藏状态在时间 t 被计算为

$$h_t = f(h_{t-1}, y_{t-1}, c, y_1, \dots, y_{t-1}),$$

类似地，下一个符号的条件分布是

$$p(y_t | y_1, \dots, y_{t-1}, c) = g(h_{t-1}, y_{t-1}, c).$$

为了生成激活函数 f 和 g (后者必须产生有效的概率，例如用一个softmax函数)，

图1: 所提出的RNN编码器-解码器架构的图解。

所提出的RNN编码器-解码器模型被联合训练来最大化条件对数似然

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^N \log p_{\theta}(y_n | x_n),$$

其中 θ 是模型参数的集合，每一个 (x_n, y_n) 是训练集中的（输入序列，输出序列）对。在我们的情况下，作为解码器的输出，从输入开始，是可微的，我们可以使用基于梯度的算法来估计模型参数。

一旦RNN编码器-解码器训练好了，这个模型就可以以两种方式使用。一种方式是使用模型来生成给定输入序列的目标序列。另一方面，模型可以用来对给定输入和输出序列进行打分，这里的打分只是简单的概率 $p_{\theta}(y|x)$ 来自方程 (3) 和 (4)。

教学：

这个RNN编码器-解码器框架是为了处理那些输入和输出序列长度不一致的问题。例如，这在机器翻译中很常见，一个句子在一种语言中的表示长度可能和另一种语言中的不同。这个模型有两个主要组成部分：

1. **编码器：**它接收输入序列，并将其转换成一个内部的固定长度表示（也就是上文提到的 c ）。这个表示试图捕捉输入数据的全部重要信息。
2. **解码器：**它接收编码器的输出，并根据这个信息生成输出序列。在生成每个输出符号的时候，它都会考虑到当前的隐藏状态、上一个生成的符号，以及编码器的总结。

训练这样的模型涉及到调整网络的权重，以便最大化在训练数据上的条件对数似然。条件对数似然是模型输出序列的概率的对数。通过最大化这个量，模型学会生成更准确地反映给定输入序列的输出序列。

在实际应用中，编码器-解码器模型可以用来生成文本（如机器翻译或自动摘要），或用来为序列打分，比如评估一个机器翻译模型生成的翻译质量。