# Supervised Sequence Labelling with Recurrent Neural Networks - Long Short-Term Memory Networks (LSTM)

**Original paper :Supervised Sequence Labelling with Recurrent Neural Networks by Alex Graves 2012**

## 2.3 Sequence Labelling

The goal of sequence labelling is to assign sequences of labels, drawn from a fixed alphabet, to sequences of input data.

For example, one might wish to transcribe a sequence of acoustic features with spoken words (speech recognition), or a sequence of video frames with hand gestures (gesture recognition).

Although such tasks commonly arise when analysing time series, they are also found in domains with non-temporal sequences, such as protein secondary structure prediction.

For some problems the precise alignment of the labels with the input data must also be determined by the learning algorithm.

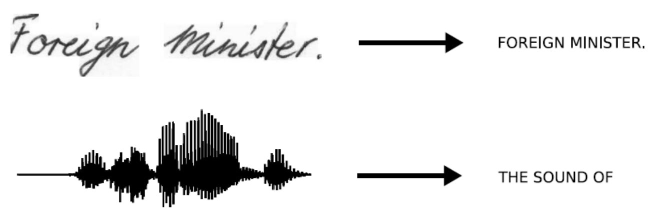


Figure 2.1: Sequence labelling.

The algorithm receives a sequence of input data, and outputs a sequence of discrete labels.

In this book however, we limit our attention to tasks where the alignment is either predetermined, by some manual or automatic preprocessing, or it is unimportant, in the sense that we require only the final sequence of labels, and not the times at which they occur.

If the sequences are assumed to be independent and identically distributed, we recover the basic framework of pattern classification, only with sequences in place of patterns (of course the data-points within each sequence are not assumed to be independent).

In practice this assumption may not be entirely justified (for example, the sequences may represent turns in a spoken dialogue, or lines of text in a handwritten form); however it is usually not too damaging as long as the sequence boundaries are sensibly chosen.

We further assume that each target sequence is at most as long as the corresponding input sequence.

With these restrictions in mind we can formalize the task of sequence labeling as follows:

Let $S$ be a set of **training examples** drawn independently from a fixed distribution

$$D_{X\times Z}$$

训练集 $S$

The **input space** $X=(R^M)^*$ is the set of all sequences of size $M$ real-valued vectors.
输入空间 $X=(R^M)^*$ 为所有向量序列的集合，序列的分量为 $M$ 维实值向量。

The **target space** $Z=L^*$ is the set of all sequences over the (finite) alphabet $L$ of labels.
目标空间 $Z=L^*$ 为所有字母序列的集合，序列的分量为有限字母表 $L$ 中的元素（字母）。

We refer to elements of $L^*$ as **label sequences** or **labellings**.
称 $L^*$ 中的元素为标记序列。

Each element (**training example**) of $S$ is a pair of sequences $(\boldsymbol{x},\boldsymbol{z})$ (From now on a bold typeface will be used to denote sequences).

$$(\boldsymbol{x},\boldsymbol{z})\in S \qquad \begin{cases}\boldsymbol{x}\in X=(R^M)^* \\ \boldsymbol{z}\in Z=L^*\end{cases}$$

The **target sequence(label sequence)** $\boldsymbol{z}=(z_1,z_2,\cdots,z_U)$ is at most as long as the **input sequence** $\boldsymbol{x}=(x_1,x_2,\cdots,x_T)$,

$$z_1,z_2,\cdots,z_U\in L \qquad x_1,x_2,\cdots,x_T\in R^M$$

i.e. $|\boldsymbol{z}|=U\le|\boldsymbol{x}|=T$

Regardless of whether the data is a time series, the distinct points (特殊点) in the input sequence are referred to as timesteps.

The task is to use $S$ to train a **sequence labelling algorithm** $h: X\mapsto Z$ to label the sequences in a test set

$$S'\subset D_{X\times Z}$$

disjoint from $S$, as accurately as possible.

In some cases we can apply additional constraints to the label sequences.

These may affect both the choice of sequence labelling algorithm and the error measures used to assess performance.

The following sections describe three classes of sequence labelling task, corresponding to progressively looser assumptions about the relationship between the input and label sequences, and discuss algorithms and error measures suitable for each. The relationship between the classes is outlined in Figure 2.2.

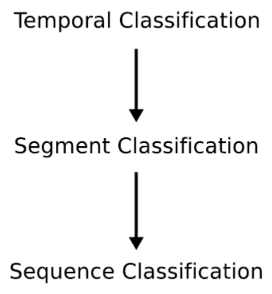


Figure 2.2: Three classes of sequence labelling task.

**Sequence classification**, where each input sequence is assigned a single class, is a special case of segment classification, where each of a predefined set of input segments is given a label.

**Segment classification** is a special case of temporal classification, where any alignment between input and label sequences is allowed.

**Temporal classification** data can be weakly labelled with nothing but the target sequences, while segment classification data must be strongly labelled with both targets and input-target alignments.

## 2.3.1 Sequence Classification(序列分类)

The most restrictive case is where the **label sequences** are constrained to be length one.

This is referred to as sequence classification, since each **input sequence** is assigned to a **single class**.

**Sequence Classification** 序列分类的目标序列（**label sequence**）长度为 1，也就是对输入序列（**input sequence**）进行简单分类，每个类有单一标记。

Examples of sequence classification task include the identification of a single spoken work and the recognition of an individual handwritten letter.

A key feature of such tasks is that the entire sequence can be processed before the classification is made.

If the input sequences are of mixed length, or can be easily padded to a mixed length, they can be collapsed into a single input vector and any of the standard pattern classification algorithms mentioned in Section 2.2 can be applied.

A prominent testbed for mixed-length sequence classification is the MNIST isolated digits dataset

(LeCun et al., 1998a).

Numerous pattern classification algorithms have been applied to MNIST, including convolutional neural networks (LeCun et al., 1998a; Simard et al., 2003) and support vector machines (LeCun et al., 1998a; Decoste and Scholkopf, 2002).

However, even if the input length is mixed, algorithm that are inherently sequential may be beneficial, since they are better able to adapt to translations and distortions in the input data.

This is the rationale behind the application of multidimensional recurrent neural networks to MNIST in Chapter 8.

As with pattern classification the obvious error measure is the percentage of misclassifications, referred to as the **sequence error rate** $E^{seq}$ in this context:

$$E^{seq}(h,S') = \frac{100}{|S'|} \sum_{(\boldsymbol{x},\boldsymbol{z}) \in S'} \begin{cases} 0 & if \quad h(\boldsymbol{x}) = \boldsymbol{z} \\ 1 & otherwise \end{cases}$$

(2.15)
Where $|S'|$ is the number of elements in $S'$.

### 2.3.2 Segment Classification(片段分类)

**Segment classification** refers to those tasks where the **target sequences** consist of **multiple labels**, but the locations of the labels---that is, the positions of the **input segments** to which the labels apply---are known in advance.

**Segment classification** 片段分类的目标序列为多标记序列，标记的位置，也就是需要标记的输入片段（**input segments**）的位置，是预先已知的。

Segment classification is common in domains such as **natural language processing** and **bioinformatics**, where the inputs are discrete and can be trivially segmented.

It can also occur in domains where segmentation is difficult, such as **audio or image processing**; however this typically requires hand-segmented training data, which is difficult to obtain.

A crucial element of segment classification, missing from sequence classification, is the use of **context information** (上下文信息) from either side of the segments to be classified.

The effective use of context is vital to the success of segment classification algorithms, as illustrated in Figure 2.3.

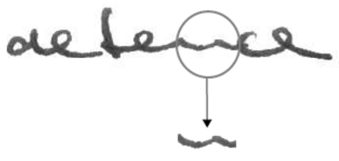


Figure 2.3: Importance of context in segment classification.

The word 'defence' is clearly legible(易读的). However the letter 'n' in isolation is ambiguous.

This presents a problem for standard pattern classification algorithms, which are designed to process only one input at a time.

A simple solution is to collect the data on either side of the segments into **time-windows**, and use the windows as input patterns.

However as well as the aforementioned issue of shifted or distorted data, the time-window approach suffers from the fact that the range of useful context (and therefore the required time-window size) is generally unknown, and may vary from segment to segment.

Consequently the case for sequential algorithms is stronger here than in sequence classification.

The obvious error measure for segment classification is the **segment error rate** $E^{seg}$, which simply counts the percentage of misclassified segments.

$$E^{seg}(h,S') = \frac{1}{Z} \sum_{(\boldsymbol{x},\boldsymbol{z})\in S'} HD(h(\boldsymbol{x}),\boldsymbol{z})$$

(2.16)
Where

$$Z = \sum_{(\boldsymbol{x},\boldsymbol{z})\in S'} |\boldsymbol{z}|$$

(2.17)
and $HD(\boldsymbol{p},\boldsymbol{q})$ is the hamming distance between two equal length sequences $\boldsymbol{p}$ and $\boldsymbol{q}$ (i.e. the number of places in which they differ).

In speech recognition, the **phonetic classification** of each **acoustic frame** as a separate segment is often known as **framewise phoneme classification**.

In this context the segment error rate is usually referred to as the **frame error rate**.

Various neural network architectures are applied to framewise phoneme classification in Chapter 5.

In image processing, the classification of each pixel, or block of pixels, as a separate segment is known as image segmentation.

Multidimensional recurrent neural networks are applied to image segmentation in Chapter 8.

### 2.3.3 Temporal Classification(时序分类)

In the most general case, nothing can be assumed about the label sequences except that their length is less than or equal to that of the input sequences.

They may even be empty.

We refer to this situation as **temporal classification** (Kadous, 2002).

The key distinction between **temporal classification** and **segment classification** is that the former requires an algorithm that can decide where in the **input sequence** the classifications should be made.

**temporal classification** 与 **segment classification** 的关键区别是：**temporal classification** 要有一个算法来确定输入序列（input sequence）的哪些部分需要进行分类。对 **segment classification** 而言，需要标记的位置是已知的。

典型例子：连续语音识别 continuous speech recognition

This in turn requires an implicit or explicit model of the global structure of the sequence.

For temporal classification, the segment error rate is inapplicable, since the segment boundaries are unknown.

Instead we measure the total number of substitutions, insertions and deletions that would be required to turn one sequence into the other, giving us the **label error rate** $E^{lab}$:

$$E^{lab}(h, S') = \frac{1}{Z} \sum_{(\boldsymbol{x},\boldsymbol{z}) \in S'} ED(h(\boldsymbol{x}), \boldsymbol{z})$$

(2.18)

where $ED(\boldsymbol{p}, \boldsymbol{q})$ is the **edit distance** between the two sequences $\boldsymbol{p}$ and $\boldsymbol{q}$ (i.e. the minimum number of insertions, substitutions and deletions required to change $\boldsymbol{p}$ into $\boldsymbol{q}$). $ED(\boldsymbol{p}, \boldsymbol{q})$ can be calculated in $O(|\boldsymbol{p}|, |\boldsymbol{q}|)$ time (Navarro, 2001).

The label error rate is typically multiplied by 100 so that it can be interpreted as a percentage (a convention we will follow in this book); however, unlike the other error measures considered in this chapter, it is not a true percentage, and may give values higher than 100.

A family of similar error measures can be defined by introducing other types of edit operation, such as transpositions (caused by e.g. typing errors), or by weighting the relative importance of the operations.

For the purposes of this book however, the label error rate is sufficient.

We will usually refer to the label error rate according to the type of label in question, for example phoneme error rate or word error rate.

For some temporal classification tasks a completely correct labelling is required and the degree of error is unimportant.

In this case the sequence error rate (2.15) should be used to assess performance.

The use of hidden Markov model-recurrent neural network hybrids for temporal classification is investigated in Chapter 6, and a neural-network-only approach to temporal classification is introduced in Chapter 7.

## 3.2 Recurrent Neural Networks

In the previous section we considered feedforward neural networks whose connections did not form cycles.

If we relax this condition, and allow cyclical connections as well, we obtain recurrent neural networks (RNNs).

As with feedforward networks, many varieties of RNN have been proposed, such as Elman networks (Elman, 1990), Jordan networks (Jordan, 1990), time delay neural networks (Lang et al., 1990) and echo state networks (Jaeger, 2001).

In this chapter, we focus on a simple RNN containing a single, self connected hidden layer, as shown in Figure 3.3.

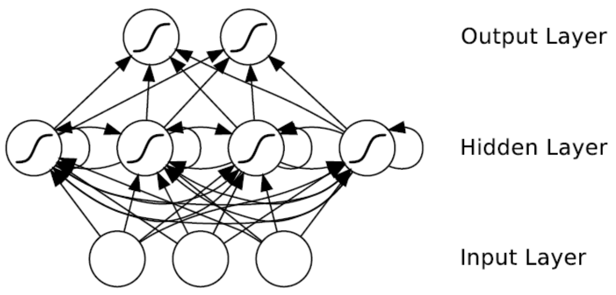


Figure3.3 A recurrent neural network

While the difference between a multilayer perceptron and an RNN may seem trivial, the implications for sequence learning are far-reaching.

An MLP can only map from input to output vectors, whereas an RNN can in principle map from the entire history of previous inputs to each output.

Indeed, the equivalent result to the universal approximation theory for MLPs is that an RNN with a sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping to arbitrary accuracy (Hammer, 2000).

The key point is that the recurrent connections allow a 'memory' of previous inputs to persist in the network's internal state, and thereby influence the network output.

RNN 网络的关键是，以前的输入以某种记忆的形式保持在网络状态中，并影响网络的输出.

### 3.2.1 Forward Pass

The forward pass of an RNN is the same as that of a multilayer perceptron with a single hidden layer, except that activations arrive at the hidden layer from both the current external input and the hidden layer activations from the previous timestep.

Consider a length $T$ input sequence **x** presented to an RNN with $I$ input units, $H$ hidden units, and $K$ output units.

$$\boldsymbol{x}^t = \begin{pmatrix} x_1^t \\ \vdots \\ x_I^t \end{pmatrix} = (x_1^t, \cdots, x_I^t)' \qquad t = 1, \cdots, T$$

Let $x_i^t$ be the value of input *i* at time *t*, and let $a_j^t$ and $b_j^t$ be respectively the network input to unit *j* at time *t* and the activation of unit *j* at time *t*. For the hidden units we have

$$a_h^t = \sum_{i=1}^{I} w_{ih} \cdot x_i^t + \sum_{h'=1}^{H} w_{h'h} \cdot b_{h'}^{t-1}$$

(3.30)

对于隐含层神经元 $h$ 其 $t$ 时刻（或称为第 $t$ 个样本）输入 $a_h^t$ 由两部分组成：

（1）$\sum_{i=1}^{I} w_{ih} \cdot x_i^t$ ，输入层输入，输入层数据 $(x_i^t)_{i=1\cdots I} = (x_1^t, \cdots, x_I^t)$ 及对应权值 $(w_{ih})_{i=1\cdots I} = (w_{1h}, \cdots, w_{Ih})$ 的内积，$I$ 为输入层维度。

（2）$\sum_{h'=1}^{H} w_{h'h} \cdot b_{h'}^{t-1}$ ，隐含层循环输入，隐含层的每个神经元 $h'$ 前一时刻 $(t-1)$ 的输出 $\left(b_{h'}^{t-1}\right)_{h'=1\cdots H} = (b_1^{t-1}, \cdots, b_H^{t-1})$ 及对应权值 $(w_{h'h})_{h'=1\cdots H} = (w_{1h}, \cdots, w_{Hh})$ 的内积，$H$ 为隐含层维度。注意，此神经元 $h$ 自己的前一时刻输出 $b_h^{t-1}$ 也在其中。

Nonlinear, differentiable activation functions are then applied exactly as for an MLP

$$b_h^t = \theta_h(a_h^t)$$

(3.31)

隐含层神经元 $h$ 其 $t$ 时刻（或称为第 $t$ 个样本）的输出 $b_h^t$ 由激活函数 $\theta_h()$ 确定。

The complete sequence of hidden activations can be calculated by starting at $t$ = 1 and recursively applying (3.30) and (3.31), incrementing $t$ at each step.

Note that this requires initial values $b_j^0$ to be chosen for the hidden units, corresponding to the network's state before it receives any information from the data sequence.

In this book, the initial values are always set to zero.

However, other researchers have found that RNN stability and performance can be improved by using nonzero initial values (Zimmermann et al., 2006a).

The network inputs to the output units can be calculated at the same time as the hidden activations:

输出层神经元 output units 的输入 network inputs 可以由隐含层的激活值 hidden activations 计算：

$$a_k^t = \sum_{h=1}^{H} w_{hk} \cdot b_h^t \qquad \begin{cases} k = 1, \cdots, K \\ t = 1, \cdots, T \end{cases}$$

(3.32)

输出层第 $k$ 个神经元 $t$ 时刻的输入 $a_k^t = \sum_{h=1}^{H} w_{hk} \cdot b_h^t$ ，隐含层激活值 $(b_h^t)_{h=1\cdots H} = (b_1^t, \cdots, b_H^t)$ 及对应权值 $(w_{hk})_{h=1\cdots H} = (w_{1k}, \cdots, w_{Hk})$ 的内积，$H$ 为隐含层维度。

For **sequence classification** and **segment classification** tasks (Section 2.3) the MLP output **activation functions** described in Section 3.1.2 (that is, logistic sigmoid for two classes and softmax for multiple classes) can be reused for RNNs, with the classification targets typically presented at the ends of the sequences or segments.

It follows that the **loss functions** in Section 3.1.3 can be reused too. **Temporal classification** is more challenging, since the locations of the target classes are unknown. Chapter 7 introduces an output layer specifically designed for temporal classification with RNNs.

### 3.2.2 Backward Pass

Given the partial derivatives of some differentiable loss function $L$ with respect to the network outputs, the next step is to determine the derivatives with respect to the weights.

Two well-known algorithms have been devised to efficiently calculate weight derivatives for RNNs: **real time recurrent learning**(**RTRL**; Robinson and Fallside, 1987) and **backpropagation through time**(**BPTT**; Williams and Zipser, 1995; Werbos, 1990).

We focus on BPTT since it is both conceptually simpler and more efficient in computation time (though not in memory).

Like standard backpropagation, BPTT consists of a repeated application of the chain rule.
BPTT（backpropagation through time）算法重复使用链式法则。

The subtlety is that, for recurrent networks, the **loss function** depends on the activation of the hidden layer **not only** through its influence on the output layer, **but also** through its influence on the hidden layer at the next timestep.

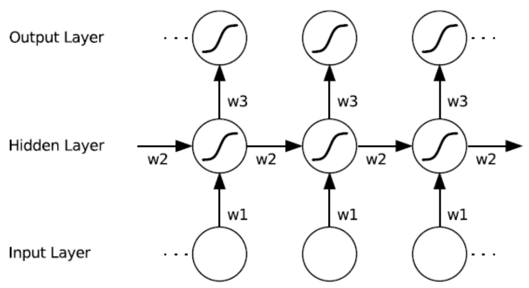


Figure 3.4: An unfolded recurrent network.

Each node represents a layer of network units at a single timestep.

The weighted connections from the input layer to hidden layer are labelled 'w1', those from the hidden layer to itself (i.e. the recurrent weights) are labelled 'w2' and the hidden to output weights are labelled 'w3'.

Note that the same weights are **reused** at every timestep. Bias weights are omitted for clarity.

Therefore，某**隐含层神经元** $h$ 在 $t$ 时刻的敏感值 $\delta_h^t$

$$\delta_h^t = \frac{\partial L}{\partial a_h^t} = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t \cdot w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} \cdot w_{hh'}\right)$$

(3.33)
$a_h^t$ 为该神经元（$h$）激活函数 $\theta(\cdot)$ 在 $t$ 时刻的输入。

$\sum_{k=1}^{K} \delta_k^t \cdot w_{hk}$ ，输出层各个神经元在 $t$ 时刻敏感值 $(\delta_k^t)_{k=1\cdots K} = (\delta_1^t, \cdots, \delta_K^t)$ 及对应隐含层神经元（$h$）到输出层各个神经元的权值 $(w_{hk})_{k=1\cdots K} = (w_{h1}, \cdots, w_{hK})$ 的内积，$K$ 为输出层维度。

$\sum_{h'=1}^{H} \delta_{h'}^{t+1} \cdot w_{hh'}$ ，隐含层各个神经元在 $t+1$ 时刻敏感值 $\left(\delta_{h'}^{t+1}\right)_{h'=1\cdots H} = (\delta_1^{t+1}, \cdots, \delta_H^{t+1})$ 及对应隐含层神经元（$h$）到隐含层各个神经元的权值 $(w_{hh'})_{h'=1\cdots H} = (w_{h1}, \cdots, w_{hH})$ 的内积，$H$ 为隐含层维度。

对输出层神经元 $k$ 在 $t$ 时刻的敏感值 $\delta_k^t$ ，可以采用 softmax 算法来确定，

$$\delta_k^t = \frac{\partial L}{\partial a_k^t}$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} \cdot b_h^t \qquad \begin{cases} k = 1, \cdots, K \\ t = 1, \cdots, T \end{cases}$$

一般而言

$$\delta_j^t \overset{\text{def}}{=} \frac{\partial L}{\partial a_j^t}$$

(3.34)
某神经元 $j$ 在 $t$ 时刻的**敏感值** $\delta_j^t$ 仍然定义为**代价函数** $L$ 相对于该神经元在 $t$ 时刻的输入值 $a_j^t$ 的导数 $\frac{\partial L}{\partial a_j^t}$。

The complete sequence of $\delta$ terms can be calculated by starting at $t$ = $T$ and recursively applying (3.33), decrementing $t$ at each step.

Note that $\delta_j^{T+1} = 0 \ \ \forall j$ , since no error is received from beyond the end of the sequence.

**敏感值** $\delta_j^t$ 计算公式为递推式公式，从 $t = T$ 开始到 $t = 1$ 结束，初始化为 $\delta_j^{T+1} = 0 \ \ \forall j$

Finally, bearing in mind that the same weights are reused at every timestep, we sum over the whole sequence to get the derivatives with respect to the network weights:

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial L}{\partial a_j^t} \cdot \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t \cdot b_i^t$$

(3.35)
注释：原文为此式，不是很准确。事实上，应该分不同层次求解。

（1）隐含层 $h$ 到输出层 $k$ 的权值梯度值 gradient 为 $\frac{\partial L}{\partial w_{hk}}$

$$\frac{\partial L}{\partial w_{hk}} = \sum_{t=1}^{T} \frac{\partial L}{\partial a_k^t} \cdot \frac{\partial a_k^t}{\partial w_{hk}} = \sum_{t=1}^{T} \delta_k^t \cdot b_h^t \qquad \begin{cases} k = 1, \cdots, K \\ h = 1, \cdots, H \end{cases}$$

$$\delta_k^t = \frac{\partial L}{\partial a_k^t} \qquad a_k^t = \sum_{h=1}^{H} w_{hk} \cdot b_h^t \qquad \begin{cases} k = 1, \cdots, K \\ t = 1, \cdots, T \end{cases}$$

$\delta_k^t$ 为输出层 $k$ 的 $t$ 时刻敏感值，$a_k^t$ 为输出层 $k$ 的激活函数的 $t$ 时刻输入值，$b_h^t$ 为隐含层 $h$ 的 $t$ 时刻输出。

（2）隐含层 $h$ 到隐含层 $h'$ 自循环的权值梯度值 gradient 为 $\frac{\partial L}{\partial w_{hh'}}$

$$\frac{\partial L}{\partial w_{hh'}} = \sum_{t=1}^{T} \frac{\partial L}{\partial a_h^t} \cdot \frac{\partial a_h^t}{\partial w_{hh'}} = \sum_{t=1}^{T} \delta_h^t \cdot b_{h'}^{t-1} \qquad \begin{cases} h = 1, \cdots, H \\ h' = 1, \cdots, H \end{cases}$$

$$\delta_h^t = \frac{\partial L}{\partial a_h^t} \qquad a_h^t = \sum_{i=1}^{I} w_{ih} \cdot x_i^t + \sum_{h'=1}^{H} w_{h'h} \cdot b_{h'}^{t-1} \qquad \begin{cases} h = 1, \cdots, H \\ t = 1, \cdots, T \end{cases}$$

$\delta_h^t$ 为隐含层 $h$ 的 $t$ 时刻敏感值，$a_h^t$ 为隐含层 $h$ 的激活函数的 $t$ 时刻输入值，$b_{h'}^{t-1}$ 为隐含层 $h'$ 的 $t-1$ 时刻输出。

（3）输入层 $i$ 到隐含层 $h$ 的权值梯度值 gradient 为 $\frac{\partial L}{\partial w_{ih}}$

$$\frac{\partial L}{\partial w_{ih}} = \sum_{t=1}^{T} \frac{\partial L}{\partial a_h^t} \cdot \frac{\partial a_h^t}{\partial w_{ih}} = \sum_{t=1}^{T} \delta_h^t \cdot x_i^t \qquad \begin{cases} h = 1, \cdots, H \\ i = 1, \cdots, I \end{cases}$$

$$\delta_h^t = \frac{\partial L}{\partial a_h^t} \qquad a_h^t = \sum_{i=1}^{I} w_{ih} \cdot x_i^t + \sum_{h'=1}^{H} w_{h'h} \cdot b_{h'}^{t-1} \qquad \begin{cases} h = 1, \cdots, H \\ t = 1, \cdots, T \end{cases}$$

$\delta_h^t$ 为隐含层 $h$ 的 $t$ 时刻敏感值，$a_h^t$ 为隐含层 $h$ 的激活函数的 $t$ 时刻输入值，$x_i^t$ 为输入层 $i$ 的 $t$ 时刻值。

## 4 Long Short-Term Memory

As discussed in the previous chapter, an important benefit of recurrent neural networks is their ability to use contextual information when mapping between input and output sequences.

Unfortunately, for standard RNN architectures, the range of context that can be in practice accessed is quite limited.

The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections.

This effect is often referred to in the literature as the **vanishing gradient problem** (Hochreiter, 1991; Hochreiteret al., 2001a; Bengio et al., 1994).

The vanishing gradient problem is illustrated schematically in Figure 4.1

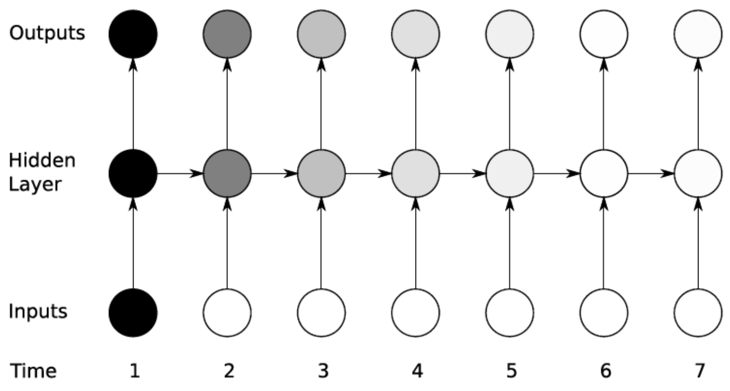


Figure 4.1: The **vanishing gradient problem** for RNNs.

The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

Numerous attempts were made in the 1990s to address the problem of vanishing gradients for RNNs.

These included non-gradient based training algorithms, such as simulated annealing and discrete error propagation (Bengio et al., 1994), explicitly introduced time delays (Lang et al., 1990; Lin et al.,1996; Plate, 1993) or time constants (Mozer, 1992), and hierarchical sequence compression (Schmidhuber, 1992).

The approach favoured by this book is the **Long Short-Term Memory (LSTM)** architecture (Hochreiter and Schmidhuber,1997).

This chapter reviews the background material for LSTM.

Section 4.1 describes the basic structure of LSTM and explains how it tackles the vanishing gradient problem.

Section 4.3 discusses an approximate and an exact algorithm for calculating the LSTM error gradient.

Section 4.4 describes some enhancements to the basic LSTM architecture.

Section 4.2 discusses the effect of preprocessing on long range dependencies.

Section 4.6 provides all the equations required to train and apply LSTM networks.

**4.1 Network Architecture**

The LSTM architecture consists of a set of recurrently connected subnets, known as **memory blocks**.

These **blocks** can be thought of as a differentiable version of the memory chips in a digital computer.

Each **block** contains one or more self-connected **memory cells** and three **multiplicative units** -- the **input**, **output** and **forget gates** -- that provide continuous analogues of **write**, **read** and **reset** operations for the cells.

每个 **block** 包含一个或多个记忆单元 **memory cells**，三个乘法控制器 **multiplicative units**，分别为输入门 **input gate**，遗忘门 **forget gate**，输出门 **output gate**。

Figure 4.2 provides an illustration of an LSTM memory block with a single cell.

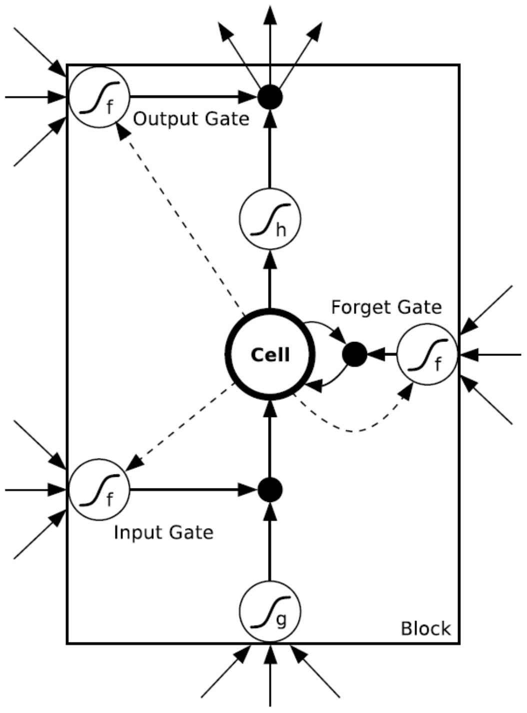


Figure 4.2: LSTM memory block with one cell (original)

The **three gates** are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles).

The **input** and **output gates** multiply the input and output of the **cell** while the **forget gate** multiplies the cell's previous state.

No activation function is applied within the **cell**.

The **gate activation function** 'f' is usually the logistic sigmoid, so that the gate activations are between 0 (gate closed) and 1 (gate open).

The **cell input** and **output activation functions** ('g' and 'h') are usually tanh or logistic sigmoid, though in some cases 'h' is the identity function.

The weighted **'peephole'** (窥视孔) connections from the cell to the gates are shown with dashed lines.

All other connections within the block are unweighted (or equivalently, have a fixed weight of 1.0).

The only **outputs from the block** to the rest of the network emanate (发出) from the value of the **output gate** multiplies the **cell output**, i.e. **block output**.

注释：**block output** 并不是只有一个值，如果一个 **block** 中有多个 **cell**，则每个 **cell** 都会对应一个 **block output**。但是一个 **block** 中的多个 **cell** 会共用一个 **output gate**。

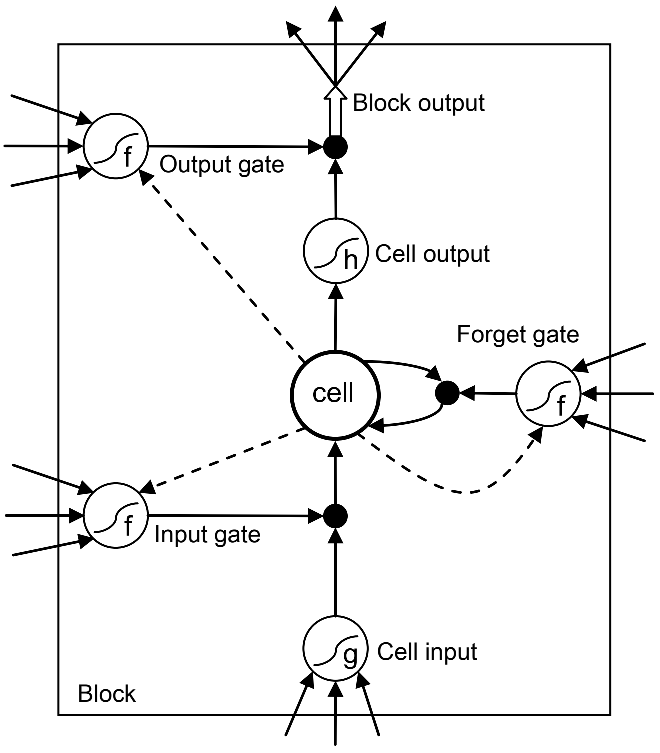


Figure 4.2: LSTM memory block with one cell (redraw)

An LSTM network is the same as a standard RNN, except that the summation units in the hidden layer are replaced by memory blocks, as illustrated in Fig. 4.3.

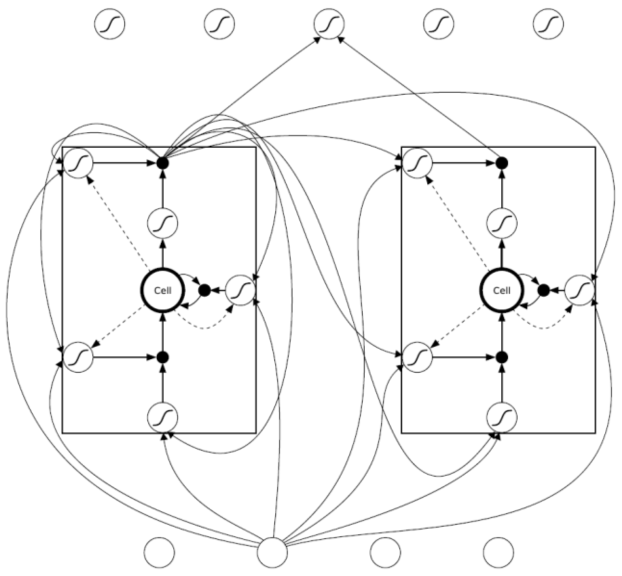


Figure 4.3: An LSTM network.

The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. Note that **each block** has **four inputs** but only **one output**.

注释：此图 Figure 4.3 中的每个 **block** 中只有一个 **cell**，所以每个 **block** 有四个输入、一个输出。如果 **block** 中多一个 **cell**，就会多一对 **cell** 的输入输出。

总结如下：
每个 **block** 是 LSTM 的基本单元，其中包含

（1）3 个控制门，为激活函数式神经元（相当于处理器），细圆圈表示，称为：
**input gate**, 输入门，输出一个乘法系数控制 **cell input** 传给 **cell** 的值。
**forget gate**, 遗忘门，输出一个乘法系数控制 **cell state** 的更新程度。
**output gate**, 输出门，输出一个乘法系数控制 **cell output** 传给 **block output** 的值。

（2）1 组存储器，包含若干个 **cell**，其中的数据值称为 **cell state**，中心粗圆圈表示。

（3）**cell** 输入器，**cell** 输出器：
**cell input**, 收集并计算准备传给 **cell** 的值。
**cell output**, **cell** 的值激活后准备输出给 **block output** 的值。
注意：**cell** 输入器，**cell** 输出器的个数取决于 **cell** 的个数。

（4）若干个数据相乘点，黑圆点表示。

（5） **block output**，该值传递给其它 **block**（也包括自己）的 **input gate**, **forget gate**, **output gate**, **cell input**, 以及输出层或下一个层次。
注意：**block output** 的个数取决于 **cell** 的个数。

（6）**block** 内的实线连线只传递数值，不含有权值（或权值恒为 1），**block** 内的虚线连线为 **peephole** 连线，附有权值。

（7）**block** 外的实线连线都附有权值。

LSTM blocks can also be mixed with ordinary summation units, although this is typically not necessary.

The same output layers can be used for LSTM networks as for standard RNNs.

The multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem.

For example, as long as the input gate remains closed (i.e. has an activation near 0), the activation of the cell will not be over written by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate.

The preservation overtime of gradient information by LSTM is illustrated in Figure 4.4.

Over the past decade, LSTM has proved successful at a range of synthetic tasks requiring long range memory, including learning context free languages(Gers and Schmidhuber, 2001), recalling high precision real numbers over extended noisy sequences (Hochreiter and Schmidhuber, 1997) and various tasks requiring precise timing and counting (Gers et al., 2002).

In particular, it has solved several artificial problems that remain impossible with any other RNN architecture.

Additionally, LSTM has been applied to various real-world problems, such as protein secondary structure prediction (Hochreiter et al., 2007; Chen and Chaudhari, 2005), music generation (Eck and Schmidhuber, 2002), reinforcement learning (Bakker, 2002), speech recognition (Graves and Schmidhuber, 2005b; Graves et al., 2006) and handwriting recognition (Liwicki et al., 2007;Graves et al., 2008).

As would be expected, its advantages are most pronounced for problems requiring the use of long range contextual information.

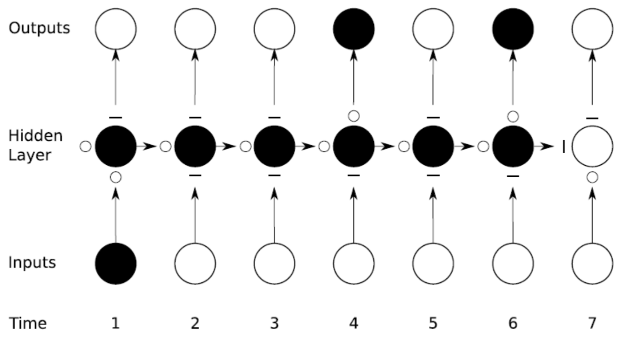


Figure 4.4: Preservation of gradient information by LSTM.

As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive.

The state of the **input**, **forget**, and **output** gates are displayed below, to the left and above the hidden layer respectively.

For simplicity, all gates are either entirely open ('O') or closed ('-'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed.

The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

## 4.2 Influence of Preprocessing

The above discussion raises an important point about the influence of preprocessing.

If we can find a way to transform a task containing long range contextual dependencies into one containing only short-range dependencies before presenting it to a sequence learning algorithm, then architectures such as LSTM become somewhat redundant.

如果有一种方法将长程相关转换为短程相关，那么 LSTM 就不是必须的。

For example, a raw speech signal typically has a sampling rate of over 40 kHz.

Clearly, a great many timesteps would have to be spanned by a sequence learning algorithm attempting to label or model an utterance(表达) presented in this form.

However when the signal is first transformed into a 100 Hz series of **mel-frequency cepstral coefficients**(梅尔频率倒谱系数), it becomes feasible to model the data using an algorithm whose contextual range is relatively short, such as a **hidden Markov model**.

隐马尔科夫模型（**HMM　hidden Markov model**）是典型的特定问题，特定设计的方法。

Nonetheless, if such a transform is difficult or unknown, or if we simply wish to get a good result without having to design task-specific preprocessing methods, algorithms capable of handling long time dependencies are essential.

问题是，这种转换方法（长程相关转换问题）是未知的、困难的，或者是领域强相关的。希望有一种一般性的、基础性的方法解决这个问题。

## 4.6 Network Equations

This section provides the equations for the activation (forward pass) and BPTT (Backpropagation Through Time) gradient calculation (backward pass) of an LSTM hidden layer within a recurrent neural network.

As before, $w_{ij}$ is the **weight** of the connection from unit *i* to unit *j*, the network **input** to unit *j* at time *t* is denoted $a_j^t$ and **activation** of unit *j* at time *t* is $b_j^t$ .

The LSTM equations are given for a single memory **block** only.

For multiple blocks the calculations are simply repeated for each block, in any order.

The subscripts $\iota$ (/aɪ'əʊtə/), $\phi$ and $\omega$ refer respectively to the **input gate**, **forget gate** and **output gate** of the block.

The subscripts *c* refers to one of the *C* **memory cells**. (There are perhaps many cells in one block.)

The **peephole weights** from cell *c* to the **input**, **forget** and **output gates** are denoted $w_{c\iota}$, $w_{c\phi}$ and $w_{c\omega}$ respectively.

$s_c^t$ is the **state** of cell *c* at time *t* (i.e. the activation of the linear cell unit).

*f* is the **activation function** of the gates, and *g* and *h* are respectively the **cell input** and **cell output activation** functions.

Let *I* be the number of inputs, *K* be the number of outputs and *H* be the number of cells in the hidden layer.

Note that only the **block outputs** $b_{c\_Bout}^{t}$ are connected to the other blocks in the layer.

The other LSTM activations, such as the states, the cell inputs, or the gate activations, are only visible within the block.

We use the **index** $h$ to refer to **block outputs** from other blocks in the hidden layer, exactly as for standard hidden units.

注释：如果有 **block** 中的 **cell** 个数多于一个，则 **index** $h$ 的总数将超过 $H$，$H$ 为隐含层 **block** 的个数。但是为了公式的简洁表达，后面公式中仍然采用 $\{h = 1, \cdots, H\}$ 的表述方式。

Unlike standard RNNs, an LSTM layer contains more inputs than outputs (because both the gates and the cells receive input from the rest of the network, but only the blocks produce output visible to the rest of the network).

We therefore define $G$ as the **total number** of inputs to the hidden layer, including cells and gates, and use the **index** $g$ to refer to these inputs when we don't wish to distinguish between the input types.

定义 $G$ 为隐含层所有输入点的个数，包含 **input gate**，**forget gate**，**output gate**，**cell input** 四种类型，当不需要区分不同输入点时，用下标 $g$ 表示。

For a standard LSTM layer with one cell per block $G$ is equal to $4H$.

对于每个隐含层 block 都只含有一个 cell 的标准 LSTM，$G = 4H$，$H$ 为隐含层 block 的个数。

As with standard RNNs the forward pass is calculated for a length $T$ input sequence **x** by starting at $t$ = 1 and recursively applying the update equations while incrementing $t$, and the BPTT backward pass is calculated by starting at $t$ = $T$, and recursively calculating the unit derivatives while decrementing $t$ to one (see Section 3.2 for details).

$$\boldsymbol{x}^t = \begin{pmatrix} x_1^t \\ \vdots \\ x_I^t \end{pmatrix} = (x_1^t, \cdots, x_I^t)' \qquad t = 1, \cdots, T$$

The final weight derivatives are found by summing over the derivatives at each timestep, as expressed in Eqn. (3.35).

Recall that

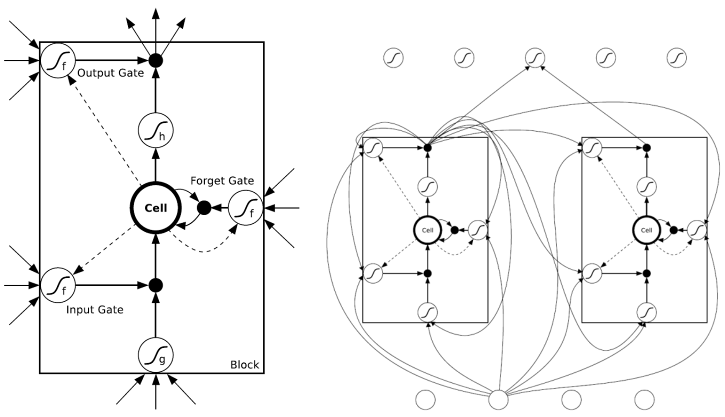
$$\delta_j^t \overset{\text{def}}{=} \frac{\partial L}{\partial a_j^t}$$

(4.1)
where $L$ is the loss function used for training.

The order in which the equations are calculated during the forward and backward passes is important, and should proceed as specified below.

As with standard RNNs, all states and activations are initialized to zero at $t = 0$, and all $\delta$ terms are zero at $t = T + 1$.

**4.6.1 Forward Pass**



**<u>Input Gate</u>**

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} \cdot x_i^t + \sum_{h=1}^{H} w_{h\iota} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} \cdot s_c^{t-1}$$

(4.2)

$$b_\iota^t = f(a_\iota^t)$$

(4.3)

每个 Block 的输入门 Input Gate 相当于一个经典神经元，不含截距，其 $t$ 时刻输入 $a_\iota^t$ 由三部分组成，其 $t$ 时刻输出 $b_\iota^t = f(a_\iota^t)$ 由激活函数决定。

（1）$\sum_{i=1}^{I} w_{i\iota} \cdot x_i^t$ ，输入层输入，输入层数据 $(x_i^t)_{i=1\cdots I} = (x_1^t, \cdots, x_I^t)$ 及对应权值 $(w_{i\iota})_{i=1\cdots I} = (w_{1\iota}, \cdots, w_{I\iota})$ 的内积，$I$ 为输入层维度。

（2）$\sum_{h=1}^{H} w_{h\iota} \cdot b_h^{t-1}$ ，隐含层循环输入，隐含层的每个 Block 前一时刻$(t-1)$的输出 $(b_h^{t-1})_{h=1\cdots H} = (b_1^{t-1}, \cdots, b_H^{t-1})$ 及对应权值 $(w_{h\iota})_{h=1\cdots H} = (w_{1\iota}, \cdots, w_{H\iota})$ 的内积，$H$ 为隐含层维度。注意，此输入门所在的自己 Block 的前一时刻输出也在其中。

（3）$\sum_{c=1}^{C} w_{c\iota} \cdot s_c^{t-1}$ ，自己 Block 的 peephole 输入，本 Block 内各个 Cell 的前一时刻 $(t-1)$ 的状态 $(s_c^{t-1})_{c=1\cdots C} = (s_1^{t-1}, \cdots, s_C^{t-1})$ 及对应的 peephole weights 权值 $(w_{c\iota})_{c=1\cdots C} = (w_{1\iota}, \cdots, w_{C\iota})$ 的内积，$C$ 为本 Block 内 Cell 的个数。

**<u>Forget Gate</u>**

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} \cdot x_i^t + \sum_{h=1}^{H} w_{h\phi} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} \cdot s_c^{t-1}$$

(4.4)

$$b_\phi^t = f\left(a_\phi^t\right)$$

(4.5)

每个 Block 的遗忘门 Forget Gate 相当于一个经典神经元，不含截距，其 $t$ 时刻输入 $a_\phi^t$ 由三部分组成，其 $t$ 时刻输出 $b_\phi^t = f\left(a_\phi^t\right)$ 由激活函数决定。

（1）$\sum_{i=1}^{I} w_{i\phi} \cdot x_i^t$ ，输入层输入，输入层数据 $(x_i^t)_{i=1\cdots I} = (x_1^t, \cdots, x_I^t)$ 及对应权值 $\left(w_{i\phi}\right)_{i=1\cdots I} = \left(w_{1\phi}, \cdots, w_{I\phi}\right)$ 的内积，$I$ 为输入层维度。

（2）$\sum_{h=1}^{H} w_{h\phi} \cdot b_h^{t-1}$ ，隐含层循环输入，隐含层的每个 Block 前一时刻$(t-1)$的输出 $(b_h^{t-1})_{h=1\cdots H} = (b_1^{t-1}, \cdots, b_H^{t-1})$ 及对应权值 $\left(w_{h\phi}\right)_{h=1\cdots H} = \left(w_{1\phi}, \cdots, w_{H\phi}\right)$ 的内积，$H$ 为隐含层维度。注意，此输入门所在的自己 Block 的前一时刻输出也在其中。

（3）$\sum_{c=1}^{C} w_{c\phi} \cdot s_c^{t-1}$ ，自己 Block 的 peephole 输入，本 Block 内各个 Cell 的前一时刻$(t-1)$的状态 $(s_c^{t-1})_{c=1\cdots C} = (s_1^{t-1}, \cdots, s_C^{t-1})$ 及对应的 peephole weights 权值 $\left(w_{c\phi}\right)_{c=1\cdots C} = \left(w_{1\phi}, \cdots, w_{C\phi}\right)$ 的内积，$C$ 为本 Block 内 Cell 的个数。

**<u>Cell input</u>**

$$a_{c\_Cin}^t = \sum_{i=1}^{I} w_{ic} \cdot x_i^t + \sum_{h=1}^{H} w_{hc} \cdot b_h^{t-1}$$

(4.6)

$$b_{c\_Cin}^t = g\left(a_{c\_Cin}^t\right)$$

(4.6.1)

每个 Block 的 Cell input 相当于一个经典神经元，不含截距，其 $t$ 时刻输入 $a_{c\_Cin}^t$ 由两部分组成，其 $t$ 时刻输出 $b_{c\_Cin}^t = g\left(a_{c\_Cin}^t\right)$ 由激活函数决定。

此处索引 $c\_Cin$ 与 **cell** 的索引 $c$ 同含义，因为 **cell input** 单元与 **cell** 单元是一一对应的。

（1）$\sum_{i=1}^{I} w_{ic} \cdot x_i^t$ ，输入层输入，输入层数据 $(x_i^t)_{i=1\cdots I} = (x_1^t, \cdots, x_I^t)$ 及对应权值 $(w_{ic})_{i=1\cdots I} = (w_{1c}, \cdots, w_{Ic})$ 的内积，$I$ 为输入层维度。

（2）$\sum_{h=1}^{H} w_{hc} \cdot b_h^{t-1}$ ，隐含层循环输入，隐含层的每个 Block 前一时刻$(t-1)$的输出 $(b_h^{t-1})_{h=1\cdots H} = (b_1^{t-1}, \cdots, b_H^{t-1})$ 及对应权值 $(w_{hc})_{h=1\cdots H} = (w_{1c}, \cdots, w_{Hc})$ 的内积，$H$ 为隐含层维度。注意，此 Cell input 所在的自己 Block 的前一时刻输出也在其中。

**<u>Cell state</u>**

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot g\left(a_{c\_Cin}^t\right)$$

(4.7)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot b_{c\_Cin}^t$$

(4.7.1)

每个 Block 的 Cell state 相当于一个存储单元，$s_c^t$，每个时刻都根据前一时刻的存储值 $s_c^{t-1}$ 及当前输入激活值 $b_{c\_Cin}^t = g\left(a_{c\_Cin}^t\right)$ 附加 **forget gate** 和 **input gate** 给出的修正参数 $b_\phi^t$ 和 $b_\iota^t$ 进行更新。

**<u>Cell Output</u>**

$$b_{c\_Cout}^t = h\left(a_{c\_Cout}^t\right)$$

(4.7.2)

$$a_{c\_Cout}^t = s_c^t \cdot 1$$

(4.7.3)

每个 Block 的 Cell output 相当于一个经典神经元，不含截距，其 $t$ 时刻输入 $a_{c\_Cout}^t$ 实为 Cell state 的当前值直接传递而来（权值为 1），其 $t$ 时刻输出 $b_{c\_Cout}^t = h\left(a_{c\_Cout}^t\right)$ 由激活函数 $h$ 决定。

此处索引 $c\_Cout$ 与 **cell** 的索引 $c$ 同含义，因为 **cell output** 单元与 **cell** 单元是一一对应的。

**Output gate**

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} \cdot x_i^t + \sum_{h=1}^{H} w_{h\omega} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} \cdot s_c^t$$

(4.8)

$$b_\omega^t = f(a_\omega^t)$$

(4.9)

每个 Block 的输出门 Output Gate 相当于一个经典神经元，不含截距，其 $t$ 时刻输入 $a_\omega^t$ 由三部分组成，其 $t$ 时刻输出 $b_\omega^t = f(a_\omega^t)$ 由激活函数决定。

（1）$\sum_{i=1}^{I} w_{i\omega} \cdot x_i^t$ ，输入层输入，输入层数据 $(x_i^t)_{i=1\cdots I} = (x_1^t, \cdots, x_I^t)$ 及对应权值 $(w_{i\omega})_{i=1\cdots I} = (w_{1\omega}, \cdots, w_{I\omega})$ 的内积，$I$ 为输入层维度。

（2）$\sum_{h=1}^{H} w_{h\omega} \cdot b_h^{t-1}$ ，隐含层循环输入，隐含层的每个 Block 前一时刻$(t-1)$的输出 $(b_h^{t-1})_{h=1\cdots H} = (b_1^{t-1}, \cdots, b_H^{t-1})$ 及对应权值 $(w_{h\omega})_{h=1\cdots H} = (w_{1\omega}, \cdots, w_{H\omega})$ 的内积，$H$ 为隐含层维度。注意，此输入门所在的自己 Block 的前一时刻输出也在其中。

（3）$\sum_{c=1}^{C} w_{c\omega} \cdot s_c^t$ ，自己 Block 的 peephole 输入，本 Block 内各个 Cell 的当前时刻$(t)$的状态 $(s_c^t)_{c=1\cdots C} = (s_1^t, \cdots, s_C^t)$ 及对应的 peephole weights 权值 $(w_{c\omega})_{c=1\cdots C} = (w_{1\omega}, \cdots, w_{C\omega})$ 的内积，$C$ 为本 Block 内 Cell 的个数。

注意，这里的 peephole 输入可以取当前时刻，而非前一时刻。

**Block Output**

$$b_{c\_Bout}^t = b_\omega^t \cdot h(s_c^t) = b_\omega^t \cdot h\left(a_{c\_Cout}^t\right) \qquad c = 1 \cdots C$$

(4.10)

$$b_{c\_Bout}^t = b_\omega^t \cdot h(s_c^t) = b_\omega^t \cdot b_{c\_Cout}^t \qquad c = 1 \cdots C$$

(4.10.1)

Block output 为某特定 block 的最终输出，其它 block 或下一个层次的单元从这个 block 只能够得到此值，由 Cell output 的输出 $h(s_c^t) = h\left(a_{c\_Cout}^t\right)$ 结合输出门 Output Gate 给出的修正参数 $b_\omega^t$ 获得。

如果一个 block 中有多个 cell，则每个 $s_c^t$ 均对应一个 $b_{c\_Bout}^t$。

此处索引 $c\_Bout$ 与 **cell** 的索引 $c$ 同含义，因为 **block output** 单元与 **cell** 单元是一一对应的。

### 4.6.2Backward Pass(sensitivity)

**求解代价函数 $L$ 相对某个值 $a$ 的导数 $\delta_a = \frac{\partial L}{\partial a}$（敏感值）的要点是，考虑该值 $a$ 在 Forward Pass 过程中会直接传递（影响）哪些后续值。**

先给两个敏感值 sensitivity 的定义：$\epsilon_c^t$ 为 Block output 的敏感值，$\epsilon_s^t$ 为 Cell state 的敏感值，

$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial b_{c\_Bout}^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial s_c^t}$$

此处索引 $c\_Bout$ 与 **cell** 的索引 $c$ 同含义，因为 **block output** 单元与 **cell** 单元是一一对应的。

注意到：针对神经元 $b_j^t = f\left(a_j^t\right)$ 的敏感值 $\delta_j^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial a_j^t}$ ，是 $L$ 对神经元输入 $a_j^t$ 的导数。

这两个值 $\epsilon_c^t$ 和 $\epsilon_s^t$ 并非针对神经元的敏感值，而是 $L$ 对特定数值的敏感值，$b_{c\_Bout}^t$ 是 block 的输出值，$s_c^t$ 是 cell 的状态值。所以这里不用 $\delta$ 而用 $\epsilon$，换个符号，以示区别。求解原理是一样的，都是链式求导法 chain rules。

<u>**Block Outputs**</u>

$$\epsilon_c^t = \frac{\partial L}{\partial b_{c\_Bout}^t} = \sum_{k=1}^{K} w_{ck} \cdot \delta_k^t + \sum_{g=1}^{G} w_{cg} \cdot \delta_g^{t+1}$$

(4.11)
Block 的输出值 $b_{c\_Bout}^t$ 的敏感值，$b_{c\_Bout}^t$ 传递给输出层，还传递给其它隐含层的输入点。

（1）$\sum_{k=1}^{K} w_{ck} \cdot \delta_k^t$ ，输出层敏感值 $(\delta_k^t)_{k=1\cdots K} = (\delta_1^t, \cdots, \delta_K^t)$ 及对应权值 $(w_{ck})_{k=1\cdots K} = (w_{c1}, \cdots, w_{cK})$ 的内积，$K$ 为输出层维度。

输出层敏感值 $(\delta_k^t)_{k=1\cdots K} = (\delta_1^t, \cdots, \delta_K^t)$ 是 **Backward Pass** 算法的数据源头，来源于 softmax 等方法。

（2）$\sum_{g=1}^{G} w_{cg} \cdot \delta_g^{t+1}$ ，隐含层各输入点 $t+1$ 时刻的敏感值 $\left(\delta_g^{t+1}\right)_{g=1\cdots G} = (\delta_1^{t+1}, \cdots, \delta_G^{t+1})$ 及对应权值 $\left(w_{cg}\right)_{g=1\cdots G} = (w_{c1}, \cdots, w_{cG})$ 的内积。

回顾：$G$ 为隐含层所有输入点的个数，包含 **input gate**，**forget gate**，**output gate**，**cell input** 四种类型，当不需要区分不同输入点时，用下标 $g$ 表示。对于每个隐含层 block 都只含有一个 cell 的标准 LSTM，$G = 4H$，$H$ 为隐含层 block 的个数。

<u>**Output Gates**</u>

$$\delta_\omega^t = \frac{\partial L}{\partial a_\omega^t} = f'(a_\omega^t) \cdot \sum_{c=1}^{C} h(s_c^t) \cdot \epsilon_c^t$$

(4.12)

输出门神经元的敏感值，该神经元的输入为 $a_\omega^t$，输出为 $b_\omega^t$，$b_\omega^t$ 直接影响 Block 的每个输出值 $b_{c\_Bout}^t(c = 1 \cdots C)$，则该敏感值计算来源于 $\frac{\partial L}{\partial b_{c\_Bout}^t} = \epsilon_c^t$。

In fact

$$\delta_\omega^t = \frac{\partial L}{\partial a_\omega^t} = \sum_{c=1}^{C}\left(\frac{\partial L}{\partial b_{c\_Bout}^t} \cdot \frac{\partial b_{c\_Bout}^t}{\partial b_\omega^t} \cdot \frac{\partial b_\omega^t}{\partial a_\omega^t}\right) = \left(\sum_{c=1}^{C} \epsilon_c^t \cdot h(s_c^t)\right) \cdot f'(a_\omega^t)$$

where

$$b_\omega^t = f(a_\omega^t)$$

(4.9)

$$b_{c\_Bout}^t = b_\omega^t \cdot h(s_c^t) \qquad c = 1 \cdots C$$

(4.10)

**<u>Cell Outputs</u>**

$$\delta_{c\_Cout}^t = \frac{\partial L}{\partial a_{c\_Cout}^t} = b_\omega^t \cdot h'\left(a_{c\_Cout}^t\right) \cdot \epsilon_c^t = b_\omega^t \cdot h'(s_c^t) \cdot \epsilon_c^t$$

(4.12.1)

Cell output 神经元的输入 $a_{c\_Cout}^t$ 的敏感值 $\delta_{c\_Cout}^t = \frac{\partial L}{\partial a_{c\_Cout}^t}$，该神经元的输出值 $b_{c\_Cout}^t$ 直接影响 block 的输出 $b_{c\_Bout}^t$，所以该敏感值的求解来源于 $b_{c\_Bout}^t$ 的敏感值 $\frac{\partial L}{\partial b_{c\_Bout}^t} = \epsilon_c^t$。

In fact

$$\delta_{c\_Cout}^t = \frac{\partial L}{\partial a_{c\_Cout}^t} = \frac{\partial L}{\partial b_{c\_Bout}^t} \cdot \frac{\partial b_{c\_Bout}^t}{\partial b_{c\_Cout}^t} \cdot \frac{\partial b_{c\_Cout}^t}{\partial a_{c\_Cout}^t} = \epsilon_c^t \cdot b_\omega^t \cdot h'\left(a_{c\_Cout}^t\right)$$

where

$$\frac{\partial L}{\partial b_{c\_Bout}^t} = \epsilon_c^t$$

$$\frac{\partial b_{c\_Bout}^t}{\partial b_{c\_Cout}^t} = b_\omega^t$$

$$\frac{\partial b_{c\_Cout}^t}{\partial a_{c\_Cout}^t} = h'\left(a_{c\_Cout}^t\right) = h'(s_c^t)$$

Notice that (**Cell output of Forward Pass**)

$$b_{c\_Cout}^t = h\left(a_{c\_Cout}^t\right)$$

(4.7.2)

$$a_{c\_Cout}^t = s_c^t \cdot 1$$

(4.7.3)

$$b_{c\_Bout}^t = b_\omega^t \cdot h(s_c^t) = b_\omega^t \cdot b_{c\_Cout}^t = b_\omega^t \cdot h\left(a_{c\_Cout}^t\right) \qquad c = 1 \cdots C$$

(4.10)

**<u>Cell States</u>**

$$\epsilon_s^t = \frac{\partial L}{\partial s_c^t} = (b_\omega^t \cdot h'(s_c^t) \cdot \epsilon_c^t) + \left(b_\phi^{t+1} \cdot \epsilon_s^{t+1}\right) + (w_{c\iota} \cdot \delta_\iota^{t+1}) + \left(w_{c\phi} \cdot \delta_\phi^{t+1}\right) + (w_{c\omega} \cdot \delta_\omega^t)$$

(4.13)

In fact

$$\epsilon_s^t = \frac{\partial L}{\partial s_c^t} = \left(1 \cdot \delta_{c\_Cout}^t\right) + \left(b_\phi^{t+1} \cdot \epsilon_s^{t+1}\right) + \left(w_{c\iota} \cdot \delta_\iota^{t+1}\right) + \left(w_{c\phi} \cdot \delta_\phi^{t+1}\right) + \left(w_{c\omega} \cdot \delta_\omega^t\right)$$

(4.13.1)

Cell state 状态值 $s_c^t$ 有 5 个去向，分别为

(4.7.2) 第一项

$$a_{c\_Cout}^t = s_c^t \cdot 1$$

(4.7) 第一项

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot g\left(a_{c\_Cin}^t\right)$$

(4.2) 第三项

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} \cdot x_i^t + \sum_{h=1}^{H} w_{h\iota} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} \cdot s_c^{t-1}$$

(4.4) 第三项

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} \cdot x_i^t + \sum_{h=1}^{H} w_{h\phi} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} \cdot s_c^{t-1}$$

(4.8) 第三项

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} \cdot x_i^t + \sum_{h=1}^{H} w_{h\omega} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} \cdot s_c^t$$

以上各式反向链式求导即可得到(4.13.1)

$$\epsilon_s^t = \frac{\partial L}{\partial s_c^t}$$

$$= \left(\frac{\partial L}{\partial a_{c\_Cout}^t} \cdot \frac{\partial a_{c\_Cout}^t}{\partial s_c^t}\right) + \left(\frac{\partial L}{\partial s_c^{t+1}} \cdot \frac{\partial s_c^{t+1}}{\partial s_c^t}\right) + \left(\frac{\partial L}{\partial a_\iota^{t+1}} \cdot \frac{\partial a_\iota^{t+1}}{\partial s_c^t}\right) + \left(\frac{\partial L}{\partial a_\phi^{t+1}} \cdot \frac{\partial a_\phi^{t+1}}{\partial s_c^t}\right) + \left(\frac{\partial L}{\partial a_\omega^t} \cdot \frac{\partial a_\omega^t}{\partial s_c^t}\right)$$

$$= \left(\delta_{c\_Cout}^t \cdot 1\right) + \left(\epsilon_s^{t+1} \cdot b_\phi^{t+1}\right) + \left(\delta_\iota^{t+1} \cdot w_{c\iota}\right) + \left(\delta_\phi^{t+1} \cdot w_{c\phi}\right) + \left(\delta_\omega^t \cdot w_{c\omega}\right)$$

**<u>Cell Inputs</u>**

$$\delta_{c\_Cin}^t = \frac{\partial L}{\partial a_{c\_Cin}^t} = b_\iota^t \cdot g'\left(a_{c\_Cin}^t\right) \cdot \epsilon_s^t$$

(4.14)

Cell input 神经元的输入 $a_{c\_Cin}^t$ 的敏感值 $\delta_{c\_Cin}^t = \frac{\partial L}{\partial a_{c\_Cin}^t}$，该神经元的输出值 $b_{c\_Cin}^t$ 直接影响 Block 的对应 cell 的状态值 $s_c^t$，所以该敏感值计算来源于 $\frac{\partial L}{\partial s_c^t} = \epsilon_s^t$。参考前式 (4.6)，(4.6.1)，(4.7)，(4.7.1)。

$$a_{c\_Cin}^t = \sum_{i=1}^{I} w_{ic} \cdot x_i^t + \sum_{h=1}^{H} w_{hc} \cdot b_h^{t-1}$$

(4.6)

$$b_{c\_Cin}^t = g\left(a_{c\_Cin}^t\right)$$

(4.6.1)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot g\left(a_{c\_Cin}^t\right)$$

(4.7)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot b_{c\_Cin}^t$$

(4.7.1)

In fact

$$\delta_{c\_Cin}^t = \frac{\partial L}{\partial a_{c\_Cin}^t} = \frac{\partial L}{\partial b_{c\_Cin}^t} \cdot \frac{\partial b_{c\_Cin}^t}{\partial a_{c\_Cin}^t} = \frac{\partial L}{\partial s_c^t} \cdot \frac{\partial s_c^t}{\partial b_{c\_Cin}^t} \cdot \frac{\partial b_{c\_Cin}^t}{\partial a_{c\_Cin}^t} = \epsilon_s^t \cdot b_\iota^t \cdot g'\left(a_{c\_Cin}^t\right)$$

**Forget Gates**

$$\delta_\phi^t = f'\left(a_\phi^t\right) \cdot \sum_{c=1}^{C} s_c^{t-1} \cdot \epsilon_s^t$$

(4.15)

遗忘门神经元的敏感值 $\delta_\phi^t = \frac{\partial L}{\partial a_\phi^t}$，该神经元的输入为 $a_\phi^t$，输出为 $b_\phi^t$，$b_\phi^t$ 直接影响 Block 的每个 cell 的状态值 $s_c^t (c = 1 \cdots C)$，则该敏感值计算来源于 $\frac{\partial L}{\partial s_c^t} = \epsilon_s^t$。参考前式 (4.4)， (4.5)， (4.7)，(4.7.1)。

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} \cdot x_i^t + \sum_{h=1}^{H} w_{h\phi} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} \cdot s_c^{t-1}$$

(4.4)

$$b_\phi^t = f\left(a_\phi^t\right)$$

(4.5)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot g\left(a_{c\_Cin}^t\right)$$

(4.7)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot b_{c\_Cin}^t$$

(4.7.1)

In fact

$$\delta_\phi^t = \frac{\partial L}{\partial a_\phi^t} = \frac{\partial L}{\partial b_\phi^t} \cdot \frac{\partial b_\phi^t}{\partial a_\phi^t} = \left(\sum_{c=1}^{C} \frac{\partial L}{\partial s_c^t} \cdot \frac{\partial s_c^t}{\partial b_\phi^t}\right) \cdot \frac{\partial b_\phi^t}{\partial a_\phi^t} = \left(\sum_{c=1}^{C} \epsilon_s^t \cdot s_c^{t-1}\right) \cdot f'\left(a_\phi^t\right)$$

**Input Gates**

$$\delta_\iota^t = f'(a_\iota^t) \cdot \sum_{c=1}^{C} g\left(a_{c\_Cin}^t\right) \cdot \epsilon_s^t$$

(4.16)

输入门神经元的敏感值 $\delta_\iota^t = \frac{\partial L}{\partial a_\iota^t}$，该神经元的输入为 $a_\iota^t$，输出为 $b_\iota^t$，$b_\iota^t$ 直接影响 Block 的每个 cell 的状态值 $s_c^t (c = 1 \cdots C)$，则该敏感值计算来源于 $\frac{\partial L}{\partial s_c^t} = \epsilon_s^t$。参考前式 (4.2)， (4.3)， (4.7) ，

(4.7.1)。

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} \cdot x_i^t + \sum_{h=1}^{H} w_{h\iota} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} \cdot s_c^{t-1}$$

(4.2)

$$b_\iota^t = f(a_\iota^t)$$

(4.3)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot g\left(a_{c\_Cin}^t\right)$$

(4.7)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot b_{c\_Cin}^t$$

(4.7.1)
In fact

$$\delta_\iota^t = \frac{\partial L}{\partial a_\iota^t} = \frac{\partial L}{\partial b_\iota^t} \cdot \frac{\partial b_\iota^t}{\partial a_\iota^t} = \left(\sum_{c=1}^{C} \frac{\partial L}{\partial s_c^t} \cdot \frac{\partial s_c^t}{\partial b_\iota^t}\right) \cdot \frac{\partial b_\iota^t}{\partial a_\iota^t} = \left(\sum_{c=1}^{C} \epsilon_s^t \cdot g\left(a_{c\_Cin}^t\right)\right) \cdot f'(a_\iota^t)$$

公式表如下：

**4.6.1 Forward Pass**

**<u>Input Gate</u>**

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} \cdot x_i^t + \sum_{h=1}^{H} w_{h\iota} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} \cdot s_c^{t-1}$$

(4.2)

$$b_\iota^t = f(a_\iota^t)$$

(4.3)

**<u>Forget Gate</u>**

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} \cdot x_i^t + \sum_{h=1}^{H} w_{h\phi} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} \cdot s_c^{t-1}$$

(4.4)

$$b_\phi^t = f\left(a_\phi^t\right)$$

(4.5)

**<u>Cell input</u>**

$$a_{c\_Cin}^t = \sum_{i=1}^{I} w_{ic} \cdot x_i^t + \sum_{h=1}^{H} w_{hc} \cdot b_h^{t-1}$$

(4.6)

$$b_{c\_Cin}^t = g\left(a_{c\_Cin}^t\right)$$

(4.6.1)

**<u>Cell state</u>**

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot g\left(a_{c\_Cin}^t\right)$$

(4.7)

$$s_c^t = b_\phi^t \cdot s_c^{t-1} + b_\iota^t \cdot b_{c\_Cin}^t$$

(4.7.1)

**<u>Cell Output</u>**

$$b_{c\_Cout}^t = h\left(a_{c\_Cout}^t\right)$$

(4.7.2)

$$a_{c\_Cout}^t = s_c^t \cdot 1$$

(4.7.3)

**<u>Output gate</u>**

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} \cdot x_i^t + \sum_{h=1}^{H} w_{h\omega} \cdot b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} \cdot s_c^t$$

(4.8)

$$b_\omega^t = f(a_\omega^t)$$

(4.9)

**<u>Block Output</u>**

$$b_{c\_Bout}^t = b_\omega^t \cdot h(s_c^t) = b_\omega^t \cdot h\left(a_{c\_Cout}^t\right) \qquad c = 1 \cdots C$$

(4.10)

$$b_{c\_Bout}^t = b_\omega^t \cdot h(s_c^t) = b_\omega^t \cdot b_{c\_Cout}^t \qquad c = 1 \cdots C$$

(4.10.1)

**4.6.2Backward Pass(sensitivity)**

$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial b_{c\_Bout}^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial s_c^t}$$

**<u>Block Outputs</u>**

$$\epsilon_c^t = \frac{\partial L}{\partial b_{c\_Bout}^t} = \sum_{k=1}^{K} w_{ck} \cdot \delta_k^t + \sum_{g=1}^{G} w_{cg} \cdot \delta_g^{t+1}$$

(4.11)

**<u>Output Gates</u>**

$$\delta_\omega^t = \frac{\partial L}{\partial a_\omega^t} = f'(a_\omega^t) \cdot \sum_{c=1}^{C} h(s_c^t) \cdot \epsilon_c^t$$

(4.12)

**<u>Cell Outputs</u>**

$$\delta_{c\_Cout}^{t} = \frac{\partial L}{\partial a_{c\_Cout}^{t}} = b_{\omega}^{t} \cdot h'\left(a_{c\_Cout}^{t}\right) \cdot \epsilon_{c}^{t} = b_{\omega}^{t} \cdot h'(s_{c}^{t}) \cdot \epsilon_{c}^{t}$$

(4.12.1)

**Cell States**

$$\epsilon_{s}^{t} = \frac{\partial L}{\partial s_{c}^{t}} = (b_{\omega}^{t} \cdot h'(s_{c}^{t}) \cdot \epsilon_{c}^{t}) + \left(b_{\phi}^{t+1} \cdot \epsilon_{s}^{t+1}\right) + (w_{c\iota} \cdot \delta_{\iota}^{t+1}) + \left(w_{c\phi} \cdot \delta_{\phi}^{t+1}\right) + (w_{c\omega} \cdot \delta_{\omega}^{t})$$

(4.13)

**Cell Inputs**

$$\delta_{c\_Cin}^{t} = \frac{\partial L}{\partial a_{c\_Cin}^{t}} = b_{\iota}^{t} \cdot g'\left(a_{c\_Cin}^{t}\right) \cdot \epsilon_{s}^{t}$$

(4.14)

**Forget Gates**

$$\delta_{\phi}^{t} = f'\left(a_{\phi}^{t}\right) \cdot \sum_{c=1}^{C} s_{c}^{t-1} \cdot \epsilon_{s}^{t}$$

(4.15)

**Input Gates**

$$\delta_{\iota}^{t} = f'(a_{\iota}^{t}) \cdot \sum_{c=1}^{C} g\left(a_{c\_Cin}^{t}\right) \cdot \epsilon_{s}^{t}$$

(4.16)

# Supervised Sequence Labelling with Recurrent Neural Networks - Connectionist Temporal Classification (CTC)

**Original paper :Supervised Sequence Labelling with Recurrent Neural Networks by Alex Graves 2012**

**Chapter 7**
**Connectionist Temporal Classification**

This chapter introduces the **connectionist temporal classification** (CTC) output layer for recurrent neural networks (Graves et al., 2006). 连接主义的时序分类

As its name suggests, CTC was specifically designed for temporal classification tasks; that is, for **sequence labelling problems** where the alignment between the inputs and the target labels is unknown.

Unlike the hybrid approach described in the previous chapter, CTC models all aspects of the sequence with a **single neural network**, and does not require the network to be combined with a **hidden Markov model**.

It also does not require **presegmented training data**, or external post-processing to extract the label sequence from the network outputs.

Experiments on speech and handwriting recognition show that a **BLSTM network** (Bidirectional LSTM) with a **CTC output layer** is an effective sequence labeller, generally outperforming standard HMMs and HMM-neural network hybrids, as well as more recent sequence labeling algorithms such as large margin HMMs (Sha and Saul, 2006) and conditional random fields (Lafferty et al., 2001).

Section 7.1 introduces CTC and motivates its use for temporal classification tasks.

Section 7.2 defines the mapping from CTC outputs onto label sequences, Section 7.3 provides an algorithm for efficiently calculating the probability of a given label sequence, Section 7.4 derives the CTC loss function used for network training, Section 7.5 describes methods for decoding with CTC, experimental results are presented in Section 7.6, and a discussion of the differences between CTC networks and HMMs is given in Section 7.7.

## 7.1 Background

In 1994, Bourlard and Morgan identified the following reason for the failure of purely connectionist

(纯粹的连接主义 that is, neural-network based) approaches to continuous speech recognition:

There is at least one fundamental difficulty with supervised training of a **connectionist network** for continuous speech recognition: a **target function** must be defined, even though the training is done for connected speech units where the **segmentation is generally unknown**.
(Bourlard and Morgan, 1994, chap. 5)

In other words, neural networks require separate training targets for **every segment** or **timestep** in the input sequence.

This has two important consequences.

Firstly, it means that the training data must be presegmented to provide the targets.

Secondly, since the network only outputs **local classifications**, the **global aspects** of the sequence, such as the likelihood of two labels appearing consecutively, must be modelled externally.

Indeed, without some form of post-processing the final label sequence cannot reliably be inferred at all.

In Chapter 6 we showed how **RNNs** could be used for temporal classification by combining them with **HMMs** in **hybrid systems**.

However, as well as inheriting the disadvantages of HMMs (which are discussed in depth in Section 7.7), hybrid systems do not exploit the full potential of RNNs for long-range sequence modelling.

It therefore seems preferable to train RNNs directly for temporal classification tasks.

**Connectionist temporal classification** (**CTC**) achieves this by allowing the network to **make label predictions** at any point in the input sequence, so long as the overall sequence of labels is correct.

This removes the need for presegmented data, since the alignment of the labels with the input is no longer important.

Moreover, CTC directly outputs the probabilities of the complete label sequences, which means that no external post-processing is required to use the network as a temporal classifier.

Figure 7.1 illustrates the difference between CTC and framewise classification applied to a speech signal.

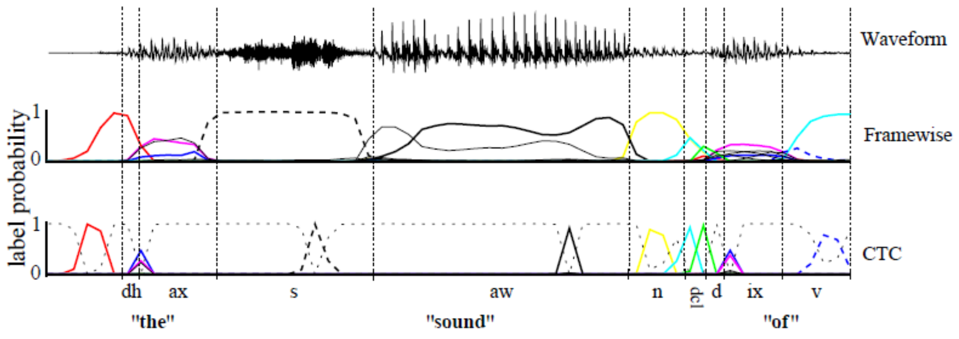


Figure 7.1: CTC and framewise classification networks applied to a speech signal. The coloured lines are the output activations, corresponding to the probabilities of observing phonemes at particular times. The CTC network predicts only the sequence of phonemes (typically as a series of spikes, separated by 'blanks', or null predictions, whose probabilities are shown as a grey dotted line), while the framewise network attempts to align them with the manual segmentation (vertical lines).

## 7.2 From Outputs to Labellings

For a **sequence labelling task** where the labels are drawn from an alphabet $A$, CTC consists of a softmax output layer (Bridle, 1990) with one more unit than there are labels in $A$.
对于序列标记问题，其标签来自于字母表 $A$，CTC 由 softmax 输出层组成，其输出单元个数比字母表 $A$ 的元素个数多一个。

The activations of **the first** $|A|$ **units** are the probabilities of outputting the corresponding labels at particular times, given the input sequence and the network weights.
前 $|\boldsymbol{A}|$ 个单元的激活值表示在给定输入序列及网络权值条件下，特定时刻（即输入序列的某位置的输入）对应的各个可能的输出标签的概率（符合度）。

The activation of the **extra unit** gives the probability of outputting a 'blank', or no label.
额外的一个单元的激活值给出“空格”（或“非标签”）的输出概率。

The complete sequence of network outputs is then used to define a distribution over all possible label sequences of length up to that of the input sequence.
网络输出的完整的序列（所有时刻的所有激活值）定义了一个针对所有可能的标签序列的分布，标签序列的长度不超过输入序列长度。

Defining the **extended alphabet** $A' = A \cup \{blank\}$, the **activation** $y_k^t$ of network output $k$ at time $t$ is interpreted as the **probability** that the network will output element $k$ of $A'$ at time $t$, given the length $T$ input sequence $\boldsymbol{x}$.
定义扩展字母表 $A' = A \cup \{blank\}$，激活值 $y_k^t$ 为 $t$ 时刻网络输出单元 $k$ 的输出值，$y_k^t$ 解释为，当给定的网络输入序列为 $\boldsymbol{x} = (x_1, x_2, \cdots, x_T)$，在 $t$ 时刻网络输出的对应标签为 $k$ 的概率（$k \in A'$），其中 $\boldsymbol{x} = (x_1, x_2, \cdots, x_T)$，$x_i$ is $a$ vector with fixed length, $i = 1, \cdots, T$

Let ${A'}^T$ denote the set of length $T$ sequences over $A'$.

$$A'^T = \{\boldsymbol{z} = (z_1, z_2, \cdots, z_T) | z_i \in A', i = 1, \cdots, T\}$$

注意到，$A'^T$ 中的序列（标签序列）的长度统一固定为 $T$，与输入序列 $\boldsymbol{x}$ 的长度是一致的。

Then, if we assume the output probabilities at each timestep to be independent of those at other timesteps (or rather, conditionally independent given $\boldsymbol{x}$), we get the following conditional distribution over $\pi \in A'^T$:

$$p(\pi|\boldsymbol{x}) = \prod_{t=1}^{T} y_{\pi_t}^t$$

(7.1)
假设输出概率对于时刻是独立的，输入序列为 $\boldsymbol{x}$，输出序列为 $\pi = (\pi_1, \pi_2, \cdots, \pi_T), \pi_i \in A', i = 1, \cdots, T$，条件概率分布为 $\{p(\pi|\boldsymbol{x}) | \pi \in A'^T\}$。

$y_{\pi_t}^t$ 表示 CTC 输出层的 $t$ 时刻输出为 $\pi_t$ 的概率，即 CTC 输出层在 $t$ 时刻与标签 $\pi_t$ 对应的输出单元的激活值。

From now on we refer to the sequences $\pi$ over $A'$ as **paths**, to distinguish them from the label sequences or labellings $\mathbf{l}$ over $A$.

前面提到过，基于字母表 $A$ 的标签序列 $\mathbf{l}$ 的长度，一般要小于输入序列 $\boldsymbol{x}$ 的长度 $T$，所以这里将基于扩展字母表 $A'$ 的固定长度的标签序列 $\pi$ 称为“路径”（**paths**）。

**注解**：当给定网络输入序列 $\boldsymbol{x} = (x_1, x_2, \cdots, x_T)$，经过网络的前馈激活，网络输出为 $\{y_k^t\}_{\begin{cases} t=1,\cdots,T \\ k \in A' = A \cup \{blank\}\end{cases}}$，实际上为一个概率阵 $\{y_k^t\}_{T \times (|A|+1)}$，考虑一个与其对应的标签符号阵：

$$\underbrace{\begin{pmatrix} y_a^1 & y_a^2 & \cdots & y_a^{T-1} & y_a^T \\ y_b^1 & y_b^2 & & y_b^{T-1} & y_b^T \\ \vdots & & \ddots & & \vdots \\ y_z^1 & y_z^2 & & y_z^{T-1} & y_z^T \\ y_\emptyset^1 & y_\emptyset^2 & \cdots & y_\emptyset^{T-1} & y_\emptyset^T \end{pmatrix}}_{T} \Leftrightarrow \underbrace{\begin{pmatrix} a & a & \cdots & a & a \\ b & b & & b & b \\ \vdots & & \ddots & & \vdots \\ z & z & & z & z \\ \emptyset & \emptyset & \cdots & \emptyset & \emptyset \end{pmatrix}}_{T}$$

假设字母表 $A' = \{a, b, c, \cdots, x, y, z, \emptyset\}$。从该阵的每一列（$i$ 列）中选取一个符号 $\pi_i \in A'$，组成一个长度为 $T$ 的序列，即为路径序列 $\pi = (\pi_1, \pi_2, \cdots, \pi_T), \pi_i \in A', i = 1, \cdots, T$，则该路径的概率为 $p(\pi|\boldsymbol{x}) = \prod_{t=1}^{T} y_{\pi_t}^t$。

The next step is to define a **many-to-one function** $: A'^T \mapsto A^{\leq T}$, from the set of paths onto the set $A^{\leq T}$ of possible labellings of $\boldsymbol{x}$ (i.e. the set of sequences of length less than or equal to $T$ over $A$).

定义一个多对一函数，$F: A'^T \mapsto A^{\leq T}$，从路径集合 $A'^T$ 映射到可能的标签集合 $A^{\leq T}$，

$$A^{\leq T} = \{\boldsymbol{z} = (z_1, z_2, \cdots, z_N) | z_i \in A, i = 1, \cdots, N, N \leq T\}$$

注意到，$A^{\leq T}$ 中的序列（标签序列）的长度不固定，为 $\leq T$，小于输入序列 $\boldsymbol{x}$ 的长度。

We do this by removing first the repeated labels and then the blanks from the paths.
此函数将第一次重复的标签字符以及空格从路径中删除。

For example

$$F(a-ab-)=F(-aa--abb)=aab$$

Intuitively, this corresponds to outputting a new label when the network either switches from predicting no label to predicting a label, or from predicting one label to another.

Since the paths are mutually exclusive, the probability of some labelling $\mathbf{l}\in A^{\leq T}$ can be calculated by summing the probabilities of all the paths mapped onto it by $F$:

$$p(\mathbf{l}|\boldsymbol{x})=\sum_{\substack{\pi\in F^{-1}(\mathbf{l}) \\ or\ F(\pi)=\mathbf{l}}} p(\pi|\boldsymbol{x})$$

(7.2)
由于路径是互斥的，则可以按照此式计算输入序列 $\boldsymbol{x}$ 条件下对应标签序列 $\mathbf{l}\in A^{\leq T}$ 的条件概率。

This '**collapsing together**' of different paths onto the same labelling is what makes it possible for CTC to use unsegmented data, because it allows the network to predict the labels without knowing in advance *where* they occur. **路径塌缩函数**

In theory, it also makes CTC networks unsuitable for tasks where the location of the labels must be determined.

However in practice CTC tends to output labels close to where they occur in the input sequence.

In Section 7.6.3 an experiment is presented in which both the labels and their approximate positions are successfully predicted by a CTC network.

### 7.2.1 Role of the Blank Labels

In the original formulation of CTC there were **no blank labels**, and $F(\pi)$ was simply $\pi$ with repeated labels removed.

This led to two problems.

Firstly, the same label could not appear twice in a row, since transitions only occurred when $\pi$ passed between different labels.

And secondly, the network was required to continue predicting one label until the next began, which is a burden in tasks where the input segments corresponding to consecutive labels are widely separated by unlabelled data (for example, in speech recognition there are often pauses or non-speech noises between the words in an utterance).

### 7.2.2 Bidirectional and Unidirectional Networks

Given that the label probabilities used for CTC are assumed to be conditioned on the entire input sequence, it seems natural to prefer a bidirectional RNN architecture.

If the network is unidirectional the label probabilities at time *t* only depend on the inputs up to *t*.

The network must therefore wait until after a given input segment is complete (or at least sufficiently complete to be identified) before emitting the corresponding label.

This returns us to the issues of past and future context discussed in Chapter 5.

Recall that for framewise classification, with a separate target for every input, one way of incorporating future context into unidirectional networks was to introduce a delay between the inputs and the targets.

Unidirectional CTC networks are in a somewhat better position, since the delay is not fixed, but can instead be chosen by the network according to the segment being labelled.

In practice the performance loss incurred by using unidirectional rather bidirectional RNNs does indeed appear to be smaller for CTC than for framewise classification.

This is worth bearing in mind for applications (such as real time speech-recognition) where bidirectional RNNs may be difficult or impossible to apply.

Figure 7.2 illustrates some of the differences between unidirectional and bidirectional CTC networks.

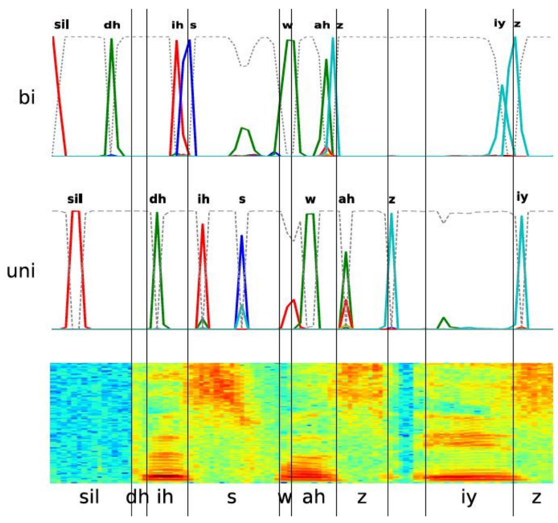


Figure 7.2: Unidirectional and Bidirectional CTC Networks Phonetically Transcribing an Excerpt from TIMIT. The spectrogram (bottom) represents the start of a TIMIT utterance, with the hand segmented

phoneme boundaries marked by vertical black lines, and the correct phonetic labels shown underneath. Output phoneme probabilities are indicated by solid coloured lines, while the dashed grey lines correspond to 'blank' probabilities. Both the unidirectional network (middle) and the bidirectional network (top) successfully label the data. However they emit the labels at different times. Whereas the unidirectional network must wait until after the corresponding segments are complete (the exceptions are 'sil' and 's', presumably because they require less context to identify) the bidirectional network may emit the labels before, after or during the segments. Another difference is that bidirectional CTC tends to 'glue together' successive labels that frequently co-occur (for example 'ah' and 'z', which combine to give the rhyming sound in 'was', 'does' and 'buzz').

## 7.3 Forward-Backward Algorithm

So far we have defined the conditional probabilities $p(\mathbf{l}|\boldsymbol{x})$ of the possible label sequences.
已经定义了输入序列 $\boldsymbol{x}$ 条件下对应标签序列 $\mathbf{l} \in A^{\leq T}$ 的条件概率 $p(\mathbf{l}|\boldsymbol{x})$。

Now we need an efficient way of calculating them.
下面要计算条件概率 $p(\mathbf{l}|\boldsymbol{x})$。

At first sight Eqn. (7.2) suggests this will be problematic: the sum is over all paths corresponding to a given labelling, and the number of these grows exponentially with the length of the input sequence (more precisely, for a **length *T* input sequence** and a **length *U* labelling**, there are $2^{T-U^2+U(T-3)}3^{(U-1)(T-U)-2}$ paths).
按照(7.2)定义式求解，路径总数是指数级的。

Fortunately the problem can be solved with a **dynamic-programming algorithm** similar to the **forward-backward algorithm** for HMMs (Rabiner, 1989).
隐马尔科夫模型 HMM 中有一个类似于动态规划算法的前向后向算法。

The key idea is that the sum over paths corresponding to a labelling $\mathbf{l}$ can be broken down into an iterative sum over paths corresponding to prefixes of that labelling.
针对所有与标签序列 $\mathbf{l}$ 对应的路径进行概率求和运算，可以分解为针对所有与标签序列 $\mathbf{l}$ 的前缀子序列对应的路径进行概率求和的迭代运算。

To allow for blanks in the output paths, we consider a modified label sequence $\mathbf{l}'$, with blanks added to the beginning and the end of $\mathbf{l}$, and inserted between every pair of consecutive labels.
为了允许输出“路径”中出现“空格”，考虑一种修订的标签序列 $\mathbf{l}'$，将标签序列 $\mathbf{l}$ 的前、后、及每一对前后标签字符之间，加入一个空格。

If the length of $\mathbf{l}$ is $U$, the length of $\mathbf{l}'$ is therefore $U' = 2U + 1$.
如果标签序列 $\mathbf{l}$ 的长度为 $U$，则修订标签序列 $\mathbf{l}'$ 的长度为 $U' = 2U + 1$。

**注释**：这个修订的标签序列 $\mathbf{l}'$ 与标签序列 $\mathbf{l}$ 是一一对应的，标签序列 $\mathbf{l}$ 不含有空格，修订标签序列 $\mathbf{l}'$ 的前后及每两个非空格符号之间有一个空格。

In calculating the probabilities of prefixes of $\mathbf{l}'$ we allow all transitions between blank and non-blank labels, and also those between any pair of distinct non-blank labels.

For a labelling $\mathbf{l}$, the **forward variable** $\alpha(t,u)$ is the summed probability of all length $t$ paths that are mapped by $F$ onto the length $u/2$ prefix of $\mathbf{l}$.
对于特定的标签序列 $\mathbf{l}$，前向变量 $\alpha(t,u)$，为所有满足 $(t,u)$ 条件的路径的概率和，这些路径长度为 $t$，而其 $F$ 映射为标签序列 $\mathbf{l}$ 的前缀的长度为 $u/2$。

**注释**：长度为 $t$ 的路径其 $F$ 映射为标签序列 $\mathbf{l}$ 的前缀的长度为 $u/2$，这一条件为间接背景条件，其实际条件为对应的修订标签序列 $\mathbf{l}'$ 的前缀的长度为 $u$，后面的递推算法的迭代参数也是 $(t,u)$，而非 $\left(t,\frac{u}{2}\right)$。也就是说，前向后向算法（递推迭代算法）的运算标的为修订的标签序列 $\mathbf{l}'$ 而非原标签序列 $\mathbf{l}$。

For some sequence $\mathbf{s}$, let $\mathbf{s}_{p:q}$ denote the subsequence $\mathbf{s}_p, \mathbf{s}_{p+1}, \cdots, \mathbf{s}_{q-1}, \mathbf{s}_q$, and let the set

$$V(t,u) = \{\pi \in A'^t : F(\pi) = \mathbf{l}_{1:u/2}, \pi_t = l'_u\}$$

$V(t,u)$ 为长度为 $t$ 的路径 $\pi \in A'^t$ 的集合，路径 $\pi$ 对应的修订标签序列 $\mathbf{l}'$ 的长度为 $u$，其压缩映射 $F(\pi) = \mathbf{l}_{1:u/2}$ 对应的标签序列前缀 $\mathbf{l}_{1:u/2}$ 的长度为 $u/2$，路径 $\pi$ 的最后的符号 $\pi_t$ 对应修订的标签序列 $\mathbf{l}'$ 的最后的符号 $l'_u$，为 $\pi_t = l'_u$。

We can then define $\alpha(t,u)$ as

$$\alpha(t,u) = \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y^i_{\pi_i}$$

(7.3)

where $u/2$ is rounded down to an integer value.

As we will see, the forward variables at time $t$ can be calculated recursively from those at time $t-1$.

Given the above formulation, the probability of $\mathbf{l}$ can be expressed as the sum of the forward variables with and without the final blank at time $T$.

$$p(\mathbf{l}|\boldsymbol{x}) = \alpha(T,U') + \alpha(T,U'-1)$$

(7.4)

此式(7.4)为前向算法的最后一步，综合概率 $\alpha(T,U')$ 对应路径长度为 $T$，修订标签序列 $\mathbf{l}'$ 的长度为 $U' = 2U+1$，即路径满长，修订标签序列也满长的情况，此时标签序列 $\mathbf{l}$ 长度也满长为 $U$。
综合概率 $\alpha(T,U'-1)$ 对应路径长度为 $T$，修订标签序列 $\mathbf{l}'$ 的长度为 $U' = 2U$，即路径满长，修订标签序列与满长只差最后一个空格的情况，此时标签序列 $\mathbf{l}$ 长度也满长为 $U$。

All correct paths must start with either a blank ($b$) or the first symbol in $\mathbf{l}$($l_1$), yielding the following initial conditions:

$$\alpha(1,1) = y_b^1$$

(7.5)

$$\alpha(1,2) = y_{l_1}^1$$

(7.6)

$$\alpha(1,u) = 0, \quad \forall u > 2$$

(7.7)

(7.5) (7.6) (7.7)为前向算法的初始化条件。

(7.5)表示当路径长度为 1 并且修订标签序列 $\mathbf{l}'$ 的（前缀）长度为 1 时，其对应的修订标签序列只能为一个空格，则路径起点只能为一个空格，所以有 $\alpha(1,1) = y_b^1$，即只能给出 $t = 1$ 时空格的出现概率。

(7.6)表示当路径长度为 1 并且修订标签序列 $\mathbf{l}'$ 的（前缀）长度为 2 时，此时可以确定其对应的标签序列 $\mathbf{l}$ 的第一个符号为 $l_1$，修订标签序列为一个空格加上标签序列 $\mathbf{l}$ 的第一个符号 $l_1$，则路径起点必须为第一个符号 $l_1$，所以有 $\alpha(1,2) = y_{l_1}^1$，即可以直接给出 $t = 1$ 时第一个符号 $l_1$ 的出现概率 $y_{l_1}^1$。

(7.7)表示当路径长度为 1 并且修订标签序列 $\mathbf{l}'$ 的（前缀）长度为 $u > 2$ 时，当网络输出的路径只有 $t = 1$ 一个时刻的输出，而要考虑其对应的修订标签序列 $\mathbf{l}'$ 的（前缀）长度为 3 个以上，表示路径的起点错过了标签序列 $\mathbf{l}$ 的第一个符号 $l_1$，则改路径不能经过塌缩函数映射为标签序列 $\mathbf{l}$，则有 $\alpha(1,u) = 0, \quad \forall u > 2$。

Thereafter the variables can be calculated recursively:

$$\alpha(t,u) = y_{l'_u}^t \cdot \sum_{i=f(u)}^{u} \alpha(t-1,i)$$

(7.8)

(7.8)式为递推公式，

where

$$f(u) = \begin{cases} u-1 & \text{if} \quad l'_u = \text{blank} \quad \text{or} \quad l'_{u-2} = l'_u \\ u-2 & \text{otherwise} \end{cases}$$

(7.9)

(7.9)式为路径回溯函数，图 7.3 中，如果修订标签序列 $\mathbf{l}'$ 的第 $u$ 个符号 $l'_u$ 为空格或与第 $u-2$ 个符号 $l'_{u-2}$ 相重复的符号，则(7.8)式为

$$\alpha(t,u) = y_{l'_u}^t \cdot \big(\alpha(t-1,u-1) + \alpha(t-1,u)\big)$$

如果修订标签序列 $\mathbf{l}'$ 的第 $u$ 个符号 $l'_u$ 为不是空格也不是与第 $u-2$ 个符号 $l'_{u-2}$ 相重复的符号，则(7.8)式为

$$\alpha(t,u) = y^t_{l'_u} \cdot \left(\alpha(t-1,u-2) + \alpha(t-1,u-1) + \alpha(t-1,u)\right)$$

Note that

$$\alpha(t,u) = 0 \qquad \forall u < [U' - 2(T-t) - 1] = [2U - 2(T-t)]$$

(7.10)

because these variables correspond to states for which there are not enough timesteps left to complete the sequence (the unconnected circles in the top right of Figure 7.3).

We also impose the boundary condition

$$\alpha(t,0) = 0 \quad \forall t$$

(7.11)

The **backward variables** $\beta(t,u)$ are defined as the summed probabilities of all paths starting at $t+1$ that complete $\mathbf{l}$ when appended to any path contributing to $\alpha(t,u)$.
**后向变量** $\beta(t,u)$，表示后缀路径的综合概率，与前向变量 $\alpha(t,u)$ 所对应前缀路径合在一起组成一条路径能够塌缩为完整的标签序列。

Let

$$W(t,u) = \{\pi \in A'^{T-t} \colon F(\hat{\pi} + \pi) = \mathbf{l} \ \ \forall \hat{\pi} \in V(t,u)\}$$

Then

$$\beta(t,u) = \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-t} y^{t+i}_{\pi_i}$$

(7.12)

The rules for initialization and recursion of the backward variables are as follows

$$\beta(T,U') = \beta(T,U'-1) = 1$$

(7.13)

$$\beta(T,u) = 0, \ \ \forall u < U' - 1$$

(7.14)

$$\beta(t,u) = \sum_{i=u}^{g(u)} \beta(t+1,i) \cdot y^{t+1}_{l'_i}$$

(7.15)

where

$$g(u) = \begin{cases} u+1 & \text{if} \quad l'_u = \text{blank} \quad \text{or} \quad l'_{u+2} = l'_u \\ u+2 & \text{otherwise} \end{cases}$$

(7.16)

Note that

$$\beta(t, u) = 0 \quad \forall u > 2t$$

(7.17)

as shown by the unconnected circles in the bottom left of Figure 7.3, and

$$\beta(t, U' + 1) = 0 \quad \forall t$$

(7.18)

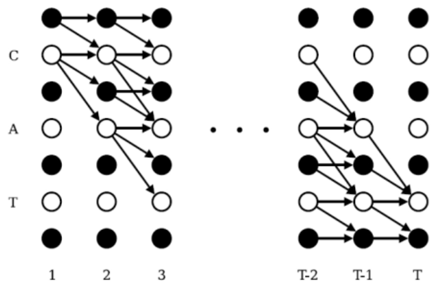


Figure 7.3: CTC forward-backward algorithm. Black circles represent blanks, and white circles represent labels. Arrows signify allowed transitions. Forward variables are updated in the direction of the arrows, and backward variables are updated against them.

对这个简单的图例而言，标签序列 $\mathbf{l} = (\mathrm{C}, \mathrm{A}, \mathrm{T})$，修订标签序列 $\mathbf{l}' = (\Phi, \mathrm{C}, \Phi, \mathrm{A}, \Phi, \mathrm{T}, \Phi)$，在图中从左到右的任何一条长度为 $T$ 的完整路径序列 $\pi = (\pi_1, \pi_2, \cdots, \pi_T), \pi_i \in \{\mathrm{C}, \mathrm{A}, \mathrm{T}, \Phi\}, i = 1, \cdots, T$，经过**路径塌缩函数** $F$ 的运算都会得到标签序列 $\mathbf{l} = (\mathrm{C}, \mathrm{A}, \mathrm{T})$，$F(\pi) = \mathbf{l}$。

**注解：前向后向算法**要解决什么问题？求出所有路径加起来的总的概率值 $p(\mathbf{l}|\boldsymbol{x})$。

图 7.3 中，黑圈表示空格，空圈表示字母，此图仅表示目标标签序列为 $\mathbf{l} = (\mathrm{C}, \mathrm{A}, \mathrm{T})$ 的情形，图中任何一条长度为 $T$ 的完整路径序列 $\pi = (\pi_1, \pi_2, \cdots, \pi_T)$，是对同样长度为 $T$ 的输入序列 $\boldsymbol{x} = (x_1, x_2, \cdots, x_T)$ 的某种网络解释（基于网络输出 $\{y_k^t\}_{\begin{cases} t=1,\cdots,T \\ k \in A' = A \cup \{blank\} \end{cases}}$，实际上为一个概率阵 $\{y_k^t\}_{T \times (|A|+1)}$，），并且每一条路径在**路径塌缩的**意义下都符合目标标签序列 $\mathbf{l}$，前向后向算法的目的是求出所有路径加起来的总的概率值 $p(\mathbf{l}|\boldsymbol{x})$。

$$p(\mathbf{l}|\boldsymbol{x}) = \sum_{\substack{\pi \in F^{-1}(\mathbf{l}) \\ or\ F(\pi) = \mathbf{l}}} p(\pi|\boldsymbol{x})$$

**注解：前向后向算法**如何求解总的概率值？

在图 7.3 中，每个黑圈、空圈称为一个节点，都对应有一个前向变量 $\alpha(t,u)$，$t$ 为节点列编号，$u$ 为节点行编号，$\alpha(t,u)$ 表示所有到达 $(t,u)$ 节点处的前缀子路径（从左上部出发的路径）的总概率，每个黑圈节点只能“向右”、“向右下”两个节点做路径转移，每个空圈节点只能“向右”、“向右下”、“向右下下”三个节点做路径转移，则每个节点的前向变量 $\alpha(t,u)$ 可以递推求得。

### 7.3.1 Log Scale

In practice, the above recursions will soon lead to underflows on any digital computer.

A good way to avoid this is to work in the log scale, and only exponentiate to find the true probabilities at the end of the calculation.

A useful equation in this context is

$$\ln(a+b) = \ln a + \ln\left(1 + e^{\ln b - \ln a}\right)$$

(7.19)

which allows the forward and backward variables to be summed while remaining in the log scale.

Note that rescaling the variables at every timestep (Rabiner, 1989) is less robust, and can fail for very long sequences.

## 7.4 Loss Function

Like the standard neural network loss functions discussed in Section 3.1.3, the CTC loss function $L(S)$ is defined as the **negative log probability** of **correctly labelling** all the training examples in some training set $S$:

$$L(S) = -\ln \prod_{(\boldsymbol{x},\boldsymbol{z})\in S} p(\boldsymbol{z}|\boldsymbol{x}) = -\sum_{(\boldsymbol{x},\boldsymbol{z})\in S} \ln p(\boldsymbol{z}|\boldsymbol{x})$$

(7.20)

Because the function is differentiable, its derivatives with respect to the network weights can be calculated with backpropagation through time (Section 3.2.2), and the network can then be trained with any gradient-based nonlinear optimisation algorithm (Section 3.3.1).

As in Chapter 2 we also define the example loss

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{z}) = -\ln p(\boldsymbol{z}|\boldsymbol{x})$$

(7.21)
概率值 $p(\boldsymbol{z}|\boldsymbol{x})$ 越大，单一代价函数 $\mathcal{L}(\boldsymbol{x},\boldsymbol{z})$ 越小。

and recall that

$$\mathcal{L}(S) = \sum_{(\boldsymbol{x},\boldsymbol{z})\in S} \mathcal{L}(\boldsymbol{x},\boldsymbol{z})$$

(7.22)

$$\frac{\partial \mathcal{L}(S)}{\partial w} = \sum_{(\boldsymbol{x},\boldsymbol{z})\in S} \frac{\partial \mathcal{L}(\boldsymbol{x},\boldsymbol{z})}{\partial w}$$

(7.23)

We now show how the algorithm of Section 7.3 can be used to calculate and differentiate $\mathcal{L}(\boldsymbol{x},\boldsymbol{z})$, and hence $\mathcal{L}(S)$.

Setting $\mathbf{l} = \mathbf{z}$ and defining the set

$$X(t,u) = \{\pi \in A'^T\colon\ F(\pi) = \mathbf{z},\ \ \pi_t = \mathbf{z}'_u\}$$

$X(t,u)$ 为长度为 $t$ 的路径 $\pi \in A'^T$ 的集合，路径 $\pi$ 对应的修订标签序列 $\mathbf{z}'$ 的长度为 $u$，其压缩映射 $F(\pi) = \mathbf{z}$，路径 $\pi$ 的最后的符号 $\pi_t$ 对应修订的标签序列 $\mathbf{z}'$ 的最后的符号 $\mathbf{z}'_u$，为 $\pi_t = \mathbf{z}'_u$。

Eqns. (7.3) and (7.12) give us

$$\alpha(t,u)\cdot\beta(t,u) = \sum_{\pi\in X(t,u)} \prod_{t=1}^{T} y^t_{\pi_t}$$

(7.24)
此式不严谨（原文如此），应该为

$$\alpha(t,u)\cdot\beta(t,u) = \sum_{\pi\in X(t,u)} \prod_{i=1}^{T} y^i_{\pi_i}$$

Substituting from (7.1) we get

$$\alpha(t,u)\cdot\beta(t,u) = \sum_{\pi\in X(t,u)} p(\pi|\mathbf{x})$$

(7.25)

From (7.2) we can see that this is the portion of the total probability $p(\mathbf{z}|\mathbf{x})$ due to those paths going through $\mathbf{z}'_u$ at time $t$.

For any $t$, we can therefore sum over all $u$ to get

$$p(\mathbf{z}|\mathbf{x}) = \sum_{u=1}^{|\mathbf{z}'|} \alpha(t,u)\cdot\beta(t,u) \qquad \forall t$$

(7.26)

注释：在图 7.3 中，任何一列（$t \in 1, \cdots, T$）上的 $\alpha(t,u) \cdot \beta(t,u)$ 总和（$u = 1, \cdots, |z'|$）是一样的。（需要证明）

Meaning that

$$\mathcal{L}(\mathbf{x},\mathbf{z}) = -\ln\left(\sum_{u=1}^{|\mathbf{z}'|} \alpha(t,u) \cdot \beta(t,u)\right) \qquad \forall t$$

(7.27)

### 7.4.1 Loss Gradient

To find the gradient of $\mathcal{L}(\mathbf{x},\mathbf{z})$, we first differentiate with respect to the network outputs $y_k^t$:

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\frac{\partial \ln p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})} \cdot \frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t}$$

(7.28)

To differentiate $p(\mathbf{z}|\mathbf{x})$ with respect to $y_k^t$, we need only consider those paths going through label $k$ at time $t$, since the network outputs do not influence each other.
为了计算 $\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t}$，只需要考虑在时刻 $t$ 经过标签 $k$ 的路径，因为网络的输出互不影响。

Noting that the same label (or blank) may occur several times in a single labelling, we define the set of positions where label $k$ occurs in $\mathbf{z}'$ as $B(\mathbf{z},k) = \{u : \mathbf{z}'_u = k\}$, which may be empty.
注意到，同一个标签在标签序列中可能出现若干次。

Observing from (7.24) that

$$\frac{\partial\big(\alpha(t,u) \cdot \beta(t,u)\big)}{\partial y_k^t} = \begin{cases} \dfrac{\alpha(t,u) \cdot \beta(t,u)}{y_k^t} & \text{if } k \text{ occurs in } \mathbf{z}' \\ 0 & \text{otherwise} \end{cases}$$

(7.29)

we can differentiate (7.26) to get

$$\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} = \frac{1}{y_k^t} \sum_{u \in B(\mathbf{z},k)} \alpha(t,u) \cdot \beta(t,u)$$

(7.30)

and substitute this into (7.28) to get

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})} \cdot \frac{1}{y_k^t} \sum_{u \in B(\mathbf{z},k)} \alpha(t,u) \cdot \beta(t,u)$$

(7.31)

Finally, to backpropagate the gradient through the output layer, we need the loss function derivatives with respect to the **outputs** $a_k^t$ before the **activation function** is applied:

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial a_k^t} = -\sum_{k'} \frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial y_{k'}^t} \cdot \frac{\partial y_{k'}^t}{\partial a_k^t}$$

(7.32)

where $k'$ ranges over all the output units.

Recalling that for **softmax** outputs

$$y_k^t = \frac{e^{a_k^t}}{\sum_{k'} e^{a_{k'}^t}}$$

$$\Rightarrow \qquad \frac{\partial y_{k'}^t}{\partial a_k^t} = y_{k'}^t(\delta_{kk'} - y_k^t)$$

(7.33)

we can substitute (7.33) and (7.31) into (7.32) to obtain

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial a_k^t} = y_k^t - \frac{1}{p(\mathbf{z}|\mathbf{x})} \sum_{u \in B(\mathbf{z},k)} \alpha(t,u) \cdot \beta(t,u)$$

(7.34)

which is the 'error signal' backpropagated through the network during training, as illustrated in Figure 7.4.

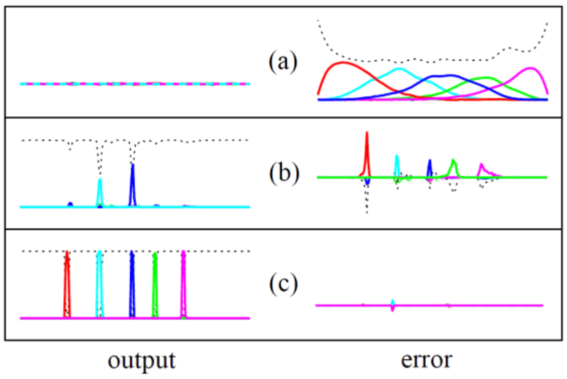


Figure 7.4: Evolution of the CTC error signal during training. The left column shows the output activations for the same sequence at various stages of training (the dashed line is the 'blank' unit); the right column shows the corresponding error signals. Errors above the horizontal axis act to increase the corresponding output activation and those below act to decrease it. (a) Initially the network has small random weights, and the error is determined by the target sequence only. (b) The network begins to make predictions and the error localises around them. (c) The network strongly predicts the correct labelling and the error virtually disappears.