Chapter 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; Hochreiter et al., 2001a; Bengio et al., 1994). The vanishing gradient problem is illustrated schematically in Figure 4.1

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,

A. Graves: Supervised Sequence Labell. with Recur. Neur. Networks, SCI 385, pp. 37–45.
springerlink.com

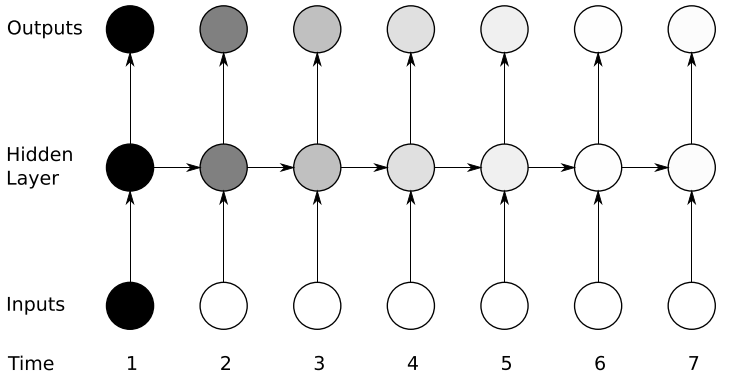

Fig. 4.1 The vanishing gradient problem for RNNs. The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

output and forget gates—that provide continuous analogues of write, read and reset operations for the cells.

Figure 4.2 provides an illustration of an LSTM memory block with a single cell. 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. 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 overwritten 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 over time 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),

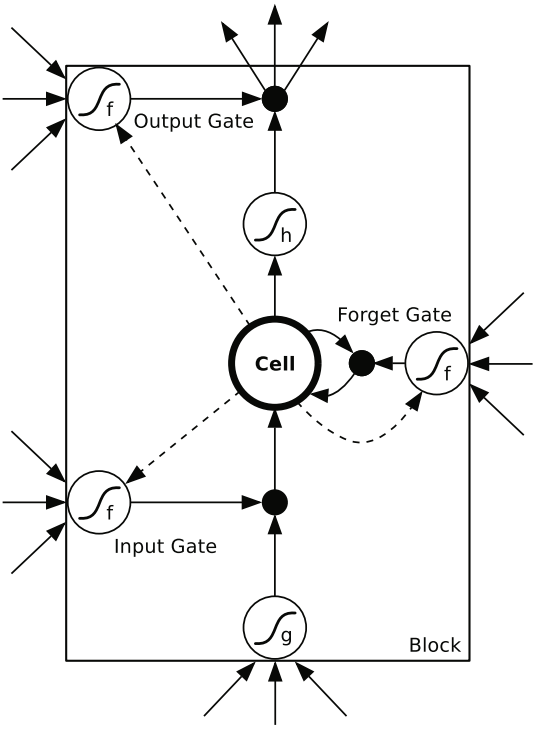

Fig. 4.2 LSTM memory block with one cell. The three gates are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles). The input and output gates multiply the input and output of the cell while 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' 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 from the output gate multiplication.

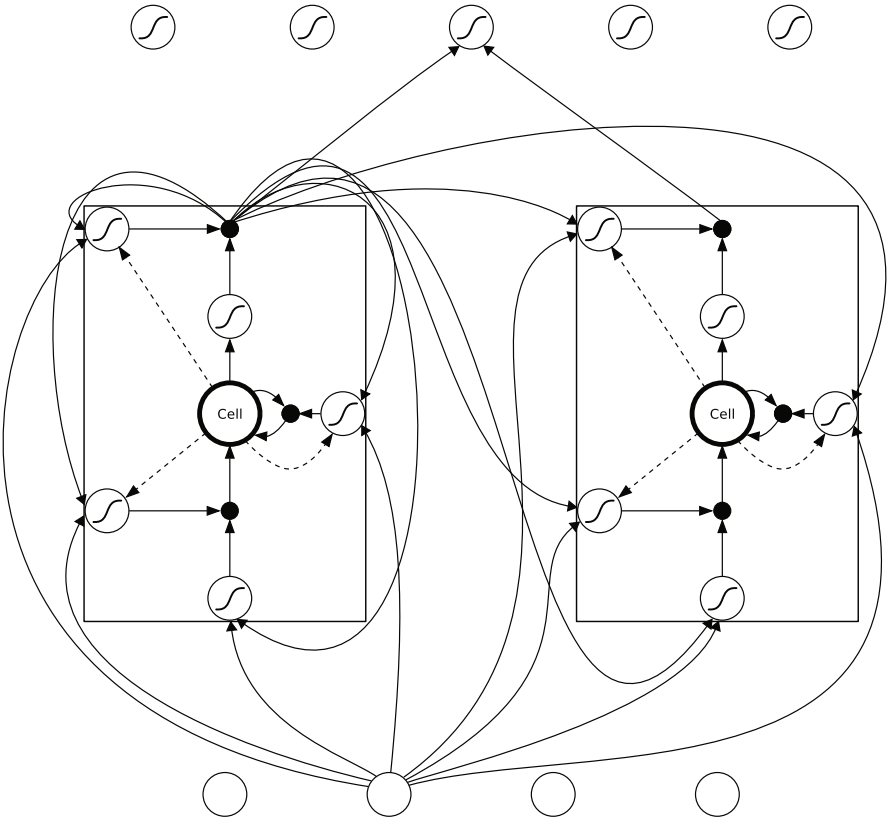

Fig. 4.3 An LSTM network. The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. Note that each block has four inputs but only one output.

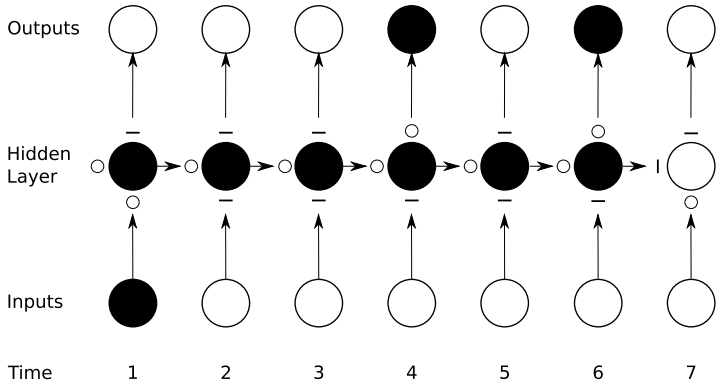

Fig. 4.4 Preservation of gradient information by LSTM. As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are 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.

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.

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. 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 presented in this form. However when the signal is first transformed into a 100 Hz series of mel-frequency cepstral coefficients, it becomes feasible to model the data using an algorithm whose contextual range is relatively short, such as a hidden Markov model.

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.

4.3 Gradient Calculation

Like the networks discussed in the last chapter, LSTM is a differentiable function approximator that is typically trained with gradient descent. Recently, non gradient-based training methods of LSTM have also been considered (Wierstra et al., 2005; Schmidhuber et al., 2007), but they are outside the scope of this book.

The original LSTM training algorithm (Hochreiter and Schmidhuber, 1997) used an approximate error gradient calculated with a combination of Real Time Recurrent Learning (RTRL; Robinson and Fallside, 1987) and Backpropagation Through Time (BPTT; Williams and Zipser, 1995). The BPTT part was truncated after one timestep, because it was felt that long time dependencies would be dealt with by the memory blocks, and not by the (vanishing) flow of activation around the recurrent connections. Truncating the gradient has the benefit of making the algorithm completely online, in the sense that weight updates can be made after every timestep. This is an important property for tasks such as continuous control or time-series prediction.

However, it is also possible to calculate the exact LSTM gradient with untruncated BPTT (Graves and Schmidhuber, 2005b). As well as being more accurate than the truncated gradient, the exact gradient has the advantage of being easier to debug, since it can be checked numerically using the technique described in Section 3.1.4.1. Only the exact gradient is used in this book, and the equations for it are provided in Section 4.6.

4.4 Architectural Variants

In its original form, LSTM contained only input and output gates. The forget gates (Gers et al., 2000), along with additional peephole weights (Gers et al., 2002) connecting the gates to the memory cell were added later to give *extended LSTM* (Gers, 2001). The purpose of the forget gates was to provide a way for the memory cells to reset themselves, which proved important for tasks that required the network to 'forget' previous inputs. The peephole connections, meanwhile, improved the LSTM's ability to learn tasks that require precise timing and counting of the internal states.

Since LSTM is entirely composed of simple multiplication and summation units, and connections between them, it is straightforward to create further variants of the block architecture. Indeed it has been shown that alternative structures with equally good performance on toy problems such as learning context-free and context-sensitive languages can be evolved automatically (Bayer et al., 2009). However the standard extended form appears to be a good general purpose structure for sequence labelling, and is used exclusively in this book.

4.5 Bidirectional Long Short-Term Memory

Using LSTM as the network architecture in a bidirectional recurrent neural network (Section 3.2.4) yields bidirectional LSTM (Graves and Schmidhuber, 2005a,b; Chen and Chaudhari, 2005; Thireou and Reczko, 2007). Bidirectional LSTM provides access to long range context in both input directions, and will be used extensively in later chapters.

4.6 Network Equations

This section provides the equations for the activation (forward pass) and BPTT 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 ι, ϕ and ω 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. 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 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 *cell outputs* b_c^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 cell outputs from other blocks in the hidden layer, exactly as for standard hidden units. As with standard RNNs the forward pass is calculated for a length T input sequence $\mathbf{x}$ 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). The final weight derivatives are found by summing over the derivatives at each timestep, as expressed in Eqn. (3.35). Recall that

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial a_j^t} \tag{4.1}$$

where $\mathcal{L}$ 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 initialised to zero at $t = 0$, and all δ terms are zero at $t = T + 1$.

4.6.1 Forward Pass

Input Gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{4.2}$$

$$b_\iota^t = f(a_\iota^t) \tag{4.3}$$

Forget Gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{4.4}$$

$$b_\phi^t = f(a_\phi^t) \tag{4.5}$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{4.6}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{4.7}$$

Output Gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{4.8}$$

$$b_\omega^t = f(a_\omega^t) \tag{4.9}$$

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \tag{4.10}$$

4.6.2 Backward Pass

$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck} \delta_k^t + \sum_{h=1}^{H} w_{ch} \delta_h^{t+1} \tag{4.11}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^{C} h(s_c^t) \epsilon_c^t \tag{4.12}$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t \tag{4.13}$$

Cells

$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t \tag{4.14}$$

Forget Gates

$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t \tag{4.15}$$

Input Gates

$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t \tag{4.16}$$