Appendix A

Recurrent neural network training

This chapter discusses the training of recurrent neural networks, and more specifically, bi-directional recurrent neural networks. The training of conventional recurrent neural networks is not explicitly considered here, as only bi-directional recurrent neural networks were used in this dissertation. BRNNs can also simplify to conventional RNNs when the number of backward hidden neurons is set to zero. In Section A.1 a general overview of gradient descent training is given, while backpropagation through time is discussed in Section A.2. The specific equations used to train the BRNNs as used in this dissertation, are then given in Section A.3. Finally, Section A.3.6 gives the strategy used to make sure that the network has a good generalisation capability.

A.1 Gradient descent training

Gradient descent, also called steepest descent, is one of the most commonly used and simplest training algorithms for neural networks [69]. In gradient descent, the idea is to adjust the network weights in the direction of the greatest rate of decrease in error.

Initial values need to be chosen for the weights for gradient descent to function cor-

rectly. Often, and as used in this dissertation, the weights are initialised uniformly from $-\epsilon$ to ϵ, where ϵ is a value that determines the range of initial weight values.

The next step in gradient descent training involves the calculation of the gradient of the error, with respect to the weights in the network, ∇E, as

$$\nabla E = \frac{\partial E}{\partial w_{ji}}, \tag{A.1}$$

where w_{ji} is the weight to which the gradient is calculated.

The weights can then be updated by adding a fraction of the negative of the gradient to the weight, or

$$w_{ji}^{t} = w_{ji}^{t-1} - \eta \nabla E, \tag{A.2}$$

where ∇E is defined in Equation (A.1) and η is called the learning rate. Provided that η is sufficiently small, the value of the error E should steadily decrease, allowing the network to learn.

A.2 Backpropagation through time

In the previous section, it was shown that the gradient of the error with respect to the network weights, is needed for the network to be able to learn. Backpropagation provides a particularly effective way to calculate these gradients. Originally, backpropagation was developed for simple multilayer perceptrons [74, 128, 129]. Backpropagation through time is based on the ability to represent any recurrent neural network as an equivalent feedforward network, unfolded in time, with the weights shared at the different time steps [126, 127, 130, 131, 132, 133, 134, 135]. The backpropagation technique of multilayer perceptrons can then be applied to the unfolded network.

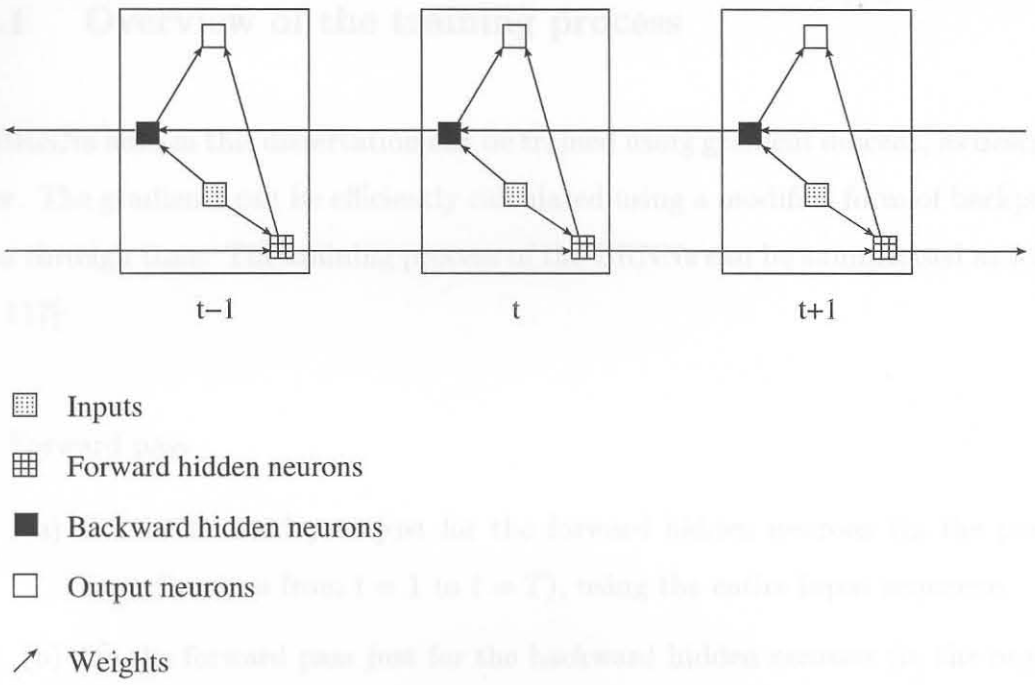

Figure A.1: Structure of the BRNN shown unfolded in time for three time steps.

Figure A.1 shows how a bi-directional recurrent neural network is unfolded for three time steps. In this figure, the arrows represent the groups of weights, and the squares the network nodes. Note that the outputs cannot be calculated until the entire sequence of forward and backward hidden node outputs have been calculated. This forward pass (consisting of three phases), and backpropagation, used to train the network, is discussed in the next section.

A.3 BRNN training equations

This section describes the training process used to train the bi-directional recurrent neural networks used in this dissertation. Since the training process is similar to that of conventional recurrent neural networks, and the fact that standard backpropagation through time needs only slight modification, the training equations will not be derived.

A.3.1 Overview of the training process

The BRNNs used in this dissertation can be trained using gradient descent, as described earlier. The gradients can be efficiently calculated using a modified form of backpropagation through time. The training process of the BRNNs can be summarised as follows [116, 117]:

1. Forward pass
   (a) Do the forward pass just for the forward hidden neurons (in the positive time direction from $t = 1$ to $t = T$), using the entire input sequence.
   (b) Do the forward pass just for the backward hidden neurons (in the negative time direction from $t = T$ to $t = 1$), using the entire input sequence.
   (c) Do the forward pass for the output neurons, using the two sequences of forward and backward hidden neuron outputs.
2. Backward pass
   (a) Do the backward pass for the output neurons, using the two sequences of output neuron outputs and target values, to obtain a sequence of delta values for each output neuron.
   (b) Do the backward pass only for the forward hidden neurons (in the negative time direction $t = T$ to $t = 1$), using the sequence of delta values of the outputs, to obtain a sequence of delta values for each forward hidden neuron.
   (c) Do the backward pass only for the backward hidden neurons (in the positive time direction $t = 1$ to $t = T$), using the sequence of delta values of the outputs, to obtain a sequence of delta values for each backward hidden neuron.
3. Calculate the gradients
4. Update the weights

A.3.2 Forward pass

Following the notation given in Chapter 2, the outputs of the forward hidden neurons can be calculated as

$$o_j^{f,t^+} = f_j^f(a_j^{f,t^+}) = f_j^f\left(\sum_{i=1}^{d} w_{ji}^{IF} x_i^{t^+} + \sum_{k=1}^{m_f} w_{jk}^{FF} o_k^{f,t^+-1} + w_j^F\right),$$
$$j = 1, 2, \ldots, m_f; t^+ = 1, 2, \ldots, T. \tag{A.3}$$

The outputs of the backward neurons are calculated as

$$o_j^{b,t^-} = f_j^b(a_j^{b,t^-}) = f_j^b\left(\sum_{i=1}^{d} w_{ji}^{IB} x_i^{t^-} + \sum_{k=1}^{m_b} w_{jk}^{BB} o_k^{b,t^-+1} + w_j^B\right),$$
$$j = 1, 2, \ldots, m_b; t^- = T, T-1, \ldots, 1. \tag{A.4}$$

The outputs of the neural network are calculated only after calculating the forward and backward neurons' outputs for the entire input sequence, $\boldsymbol{x}$, as

$$y_k^t = f_k(a_k^t) = f_k\left(\sum_{j=1}^{m_f} w_{kj}^{FO} o_j^{f,t} + \sum_{i=1}^{m_b} w_{kj}^{BO} o_j^{b,t} + w_k^O\right),$$
$$k = 1, 2, \ldots, c; t = 1, 2, \ldots, T, \tag{A.5}$$

where $o_i^{f,t}$ is the output of the i'th forward neuron at time t, $o_i^{b,t}$ is the output of the i'th backward neuron at time t, m_f is the number of forward neurons, m_b is the number of backward neurons, and f_j^f, f_j^b and f_k are the neuron transfer functions for the forward hidden, backward hidden, and output layers, respectively. In the equations above, o_j^{f,t^+-1} denotes the 1 step delayed forward hidden node output, and o_j^{b,t^-+1} is the 1 step advanced backward hidden node output. The j'th forward hidden node output at time t and t^+ is given by $o_j^{f,t}$ and o_j^{f,t^+}, respectively. The j'th backward hidden node output at time t and t^- is given by $o_j^{b,t}$ and o_j^{b,t^-}, respectively.

A.3.3 Backward pass

In order to calculate the errors, also called delta values, the errors at output of the neural network are first calculated. These errors are dependent on both the transfer functions of the neurons, as well as the performance function used. In this dissertation, the cross-entropy error function is used, which is given as

$$E = \sum_{t=1}^{T} \sum_{k=1}^{c} t_k^t ln\left(\frac{y_k^t}{t_k^t}\right), \tag{A.6}$$

where t_k^t is the target (or desired) value of the k'th output neuron at time t. The errors at the outputs can then be calculated as

$$\delta_k^t \equiv \frac{\partial E^t}{\partial a_k^t} = y_k^t - t_k^t, \qquad k = 1, 2, \ldots, c; t = 1, 2, \ldots, T, \tag{A.7}$$

where δ_k^t is the error at the k'th output at time t. The deltas at the backward neurons is calculated next, by using

$$\begin{aligned} \delta_j^{b,t^+} &= g_j^b(a_j^{b,t^+})\left(\sum_{k=1}^{c} w_{kj}^{BO}\delta_k^{b,t^+} + \sum_{k=1}^{m_b} w_{jk}^{BB}\delta_k^{b,t^+-1}\right), \\ & j = 1, 2, \ldots, m_b; t^+ = 1, 2, \ldots, T, \end{aligned} \tag{A.8}$$

where g_j^b is the derivative of the j'th backward hidden neuron transfer function. For the hyperbolic tangent transfer function given in Chapter 2, Equation (2.56), g_j is given as

$$g_j = \frac{\partial f_j}{\partial a_j} = 1 - o_j^2, \tag{A.9}$$

where o_j is the output activation of the j'th forward or backward hidden neuron. The deltas at the forward neurons can be calculated as

$$\delta_j^{f,t^-} = g_j^f(a_j^{f,t^-})\left(\sum_{k=1}^{c} w_{kj}^{FO}\delta_k^{f,t^-} + \sum_{k=1}^{m_f} w_{jk}^{FF}\delta_k^{f,t^-+1}\right),$$
$$j = 1, 2, \ldots, m_f; t^- = T, T-1, \ldots, 1. \quad \text{(A.10)}$$

A.3.4 Gradient calculation

After the forward and backward passes, sequences of neuron outputs and delta values are available for the forward hidden neurons, backward hidden neurons, and output neurons. These can then be used to calculate the gradient of the error, with respect to the network weights. The error gradients can be given as

$$\frac{\partial E}{\partial w_{jk}^{FF}} = \delta_j^{f,t} o_k^{f,t-1}, \quad j = 1, 2, \ldots, m_f, k = 1, 2, \ldots, m_f, \quad \text{(A.11)}$$

$$\frac{\partial E}{\partial w_{ji}^{IF}} = \delta_j^{f,t} x_i^t, \quad i = 1, 2, \ldots, d, j = 1, 2, \ldots, m_f, \quad \text{(A.12)}$$

$$\frac{\partial E}{\partial w_j^{F}} = \delta_j^{f,t}, \quad j = 1, 2, \ldots, m_f, \quad \text{(A.13)}$$

$$\frac{\partial E}{\partial w_{jk}^{BB}} = \delta_j^{b,t} o_k^{b,t+1}, \quad j = 1, 2, \ldots, m_b, k = 1, 2, \ldots, m_b, \quad \text{(A.14)}$$

$$\frac{\partial E}{\partial w_{ji}^{IB}} = \delta_j^{b,t} x_i^t, \quad i = 1, 2, \ldots, d, j = 1, 2, \ldots, m_b, \quad \text{(A.15)}$$

$$\frac{\partial E}{\partial w_j^{B}} = \delta_j^{b,t}, \quad j = 1, 2, \ldots, m_b, \quad \text{(A.16)}$$

$$\frac{\partial E}{\partial w_{kj}^{FO}} = \delta_k^{t} o_j^{f,t}, \quad j = 1, 2, \ldots, m_f, k = 1, 2, \ldots, c, \quad \text{(A.17)}$$

$$\frac{\partial E}{\partial w_{kj}^{BO}} = \delta_k^{t} o_j^{b,t}, \quad j = 1, 2, \ldots, m_b, k = 1, 2, \ldots, c, \quad \text{(A.18)}$$

$$\frac{\partial E}{\partial w_k^{O}} = \delta_k^{t}, \quad k = 1, 2, \ldots, c. \quad \text{(A.19)}$$

A.3.5 Update of weights

The weights of the neural network can now be updated, using the gradients of the previous section. The weight update is through the use of gradient descent. Each weight w in the network, is updated using the gradient $\frac{\partial E}{\partial w}$ as follows

$$w^t = w^{t-1} + \eta \nabla E = w^{t-1} + \eta \frac{\partial E}{\partial w}, \tag{A.20}$$

where w is one of the weights w_{ij}^{IF}, w_{ij}^{IB}, w_{jk}^{FF}, w_{kj}^{FO}, w_{kj}^{BO}, w_j^F, w_j^B, or w_k^O, and $\frac{\partial E}{\partial w}$ is one of the gradients given in the previous section, associated with w. The learning rate is denoted by η and is empirically set.

A.3.6 Generalisation capability

In order to make sure that the neural network has good generalisation, training must be stopped before overtraining occurs. In this dissertation, a simple approach to this difficult problem was taken. The training procedure can be summarised as:

1. Train the neural network on one sequence of speech vectors.
2. Evaluate the network on a small, independent, development test set and compute an average performance measure (such as the cross-entropy error).
3. Repeat steps 1 and 2 for a large number of iterations, saving the neural networks weights, and performance measure, at each iteration.
4. The neural network with the best generalisation performance is regarded as the one with the best performance on the development test set.

This training procedure is a simplification of a technique called cross-validation. The network finally chosen, is then evaluated on the full test set.