Chapter 7
Audio Recognition

Learning without thought is labor lost; thought without learning is perilous.

—Confucius

We will now deal with methods towards the actual classification or regression of audio data. A good overview on these is also found in [1].

7.1 Audio Recognition Requirements

A number of requirements speak for the consideration of diverse learning algorithms. In Table 7.1 typical such requirements are summarised.

According to these requirements, a number of learning algorithms were picked as examples in the next sections. These have proven to be reasonable choices throughout many applications as presented later in this book. They can be roughly divided into static and dynamic learners. This categorisation can best be understood by considering the chain of audio processing (cf. Chap. 4): Static learners operate on single feature vector basis (which means that multivariate time series of variable length have to be mapped to fixed size vectors), whereas their dynamic counterparts are able to handle such time series directly.

7.2 Static Learning Algorithms

7.2.1 Decision Trees

As a first learning algorithm, let us consider decision trees (DT). In principle, a DT produces a human-readable set of rules, which makes it very transparent and intuitive to understand. In case of numeric feature information, these are typically

Table 7.1 Requirements for machine learning algorithms

Requirement	Example
Adequate modeling	Static /(async.)dynamic modeling
	Data-/Knowledge-driven
	Handling of missing features
	Handling of uncertainty
	Learning stability
	Model-/Instance-based
	Transparency
Optimal accuracy	Non-linear problem handling
	Discriminative learning
	Auto-weighting of features
	Tolerance wrt. dimension
	Adaptability
	Allowance for diverse spaces
Efficiency	Real-time recognition
	Short learning/adaptation time
Economic factors	Low computational cost
	Low memory requirement
	Low HW realization costs
	Space optimization w/o training
Optimal integration	N-best provision
	Confidence provision
	Class-wise confidences
	Distributable

comparisons with constants to decide to which next comparison to branch, until the class labels are reached as terminals. A DT is thus a specific directed acyclic graph (DAG). As such, it can be defined by a set of nodes V and a set $E \subseteq V \times V$ of edges, where each element $e = (v_1, v_2) \in E$ represents a connection from node v_1 to node v_2. A path of the length P through the tree is a sequence of $v_1, \ldots, v_P,\ v_k \in V$ with $(v_k, v_{k+1}) \in E,\ k = 1, \ldots, P-1$. Starting from an undirected graph, conditions of a tree are that the graph is acyclic and connected, i.e., each node needs to be reachable by a path from each other node. By that, each tree has exactly $|V|-1$ edges. Further, there is exactly one 'root' w in the sense of a node that possesses no incoming edges, i.e., E contains no element of the form $(v, r), v \in V$. The 'leaves' are the nodes b without an outgoing edge, i.e., for which in E there exists no (b, v) with $v \in V$. All remaining nodes are referred to as 'inner' nodes [1, 2].

In the learning process, features are assigned to the inner nodes: Given a feature space of the dimension N a mapping

$$a : V \to \{1, \ldots, N\}$$

is defined. In this process, the edges are assigned the values on which the branch decisions are based upon. The values of the features as seen in the training are quantised to J_n values per feature n to reach a finite number of edges. Each inner

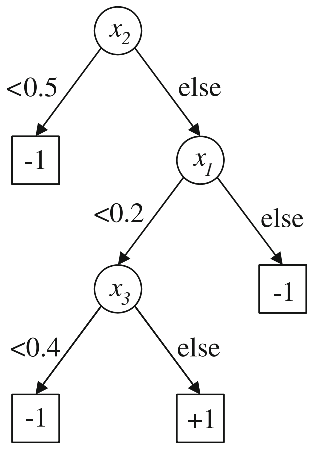


Fig. 7.1 Exemplary DT: A two-class problem is shown with three features. Circles represent the root and inner nodes, rectangles represent the leaves with the class labels

node v then has $J_{a(v)}$ outgoing edges. The leaves are assigned the according class labels.

In the recognition phase of an unknown pattern vector $\underline{x} = (x_1, \ldots, x_N)^T$, one starts at the root w and follows the path through the tree as follows: At each node v along the path choose the edge for which $x_{a(v)}$ is within this edge's interval until a leave is reached. The class to decide for is then this leave's class.

An example of a DT is shown in Fig. 7.1. In this example, quantisation of feature values was chosen as binary. This results in a simple threshold decision at each node.

An optimisation criterion is now to maximise the information gain in view of the correct classification and with the remaining features at each node. The Shannon entropy $H(Y)$ of the distribution of the class probabilities $(Y_1, \ldots, Y_M)$ can be used to this end:

$$H(Y_1, \ldots, Y_M) = -\sum_{i=1}^{M} Y_i \operatorname{ld}(Y_i). \tag{7.1}$$

For a training set $\mathcal{L}$ of pattern vectors $\underline{x}$ with known class attribution y, the needed average information $H(\mathcal{L})$ to assign an instance in $\mathcal{L}$ to a class $i \in \{1, \ldots, M\}$ is determined according to:

$$H(\mathcal{L}) = -\sum_{i=1}^{M} \hat{Y}_i \operatorname{ld}(\hat{Y}_i), \quad \hat{Y}_i = \frac{|\mathcal{L}_i|}{|\mathcal{L}|}, \tag{7.2}$$

where $\mathcal{L}_i$ is the set of elements in $\mathcal{L}$ with class attribution i.

In order to determine the contribution of each individual feature n to the aimed at class assignment, for each n the set $\mathcal{L}$ is divided into the subsets $\mathcal{L}_{n,j}$, $j = 1, \ldots, J_n$ based on the different values of n. The remaining average information $H(\mathcal{L}|n)$ needed after observation of the feature n for the class assignment results as the weighted average of the information $H(\mathcal{L}_{n,j})$, as required to classify an element of the subset $\mathcal{L}_{n,j}$:

$$H(\mathcal{L}|n) = \sum_{j=1}^{J_n} \frac{|\mathcal{L}_{n,j}|}{|\mathcal{L}|} H(\mathcal{L}_{n,j}). \tag{7.3}$$

By this equation the IG can be defined. It describes how the entropy, i.e., the information needed for the assignment, is reduced by addition of the feature n:

$$\text{IG}(\mathcal{L}, n) = H(\mathcal{L}) - H(\mathcal{L}|n). \tag{7.4}$$

However, this definition tends to favour features with a high number of different values J_n: If all elements in $\mathcal{L}$, whose features n have the same value belong to the same class—this is in particular the case, if a feature has a different value for each element in $\mathcal{L}$—, then $H(\mathcal{L}|n)$ equals zero, and by that one obtains a maximal $\text{IG}(\mathcal{L}, n)$. By introduction of the Information Gain Ratio (IGR) this can be compensated:

$$\text{IGR}(\mathcal{L}, n) = \frac{\text{IG}(\mathcal{L}, n)}{H\left(\frac{|\mathcal{L}_{n,1}|}{|\mathcal{L}|}, \ldots, \frac{|\mathcal{L}_{n,J_n}|}{|\mathcal{L}|}\right)}. \tag{7.5}$$

The term in the denominator is called split information and is computed according to Eq. (7.1). This is the information one obtains by the described split of the set $\mathcal{L}$ according to the values of the feature n.

A popular method for the training of a DT based on a training set $\mathcal{L}$ is the iterative dichotomiser 3 (ID3) algorithm [3]. ID3 constructs the DT recursively for the overall feature set by concatenation of sub-trees for each subset of the features. For a given set of features $\mathcal{M} \subseteq \{1, \ldots, N\}$ and training set $\mathcal{L}$, the steps are as follows:

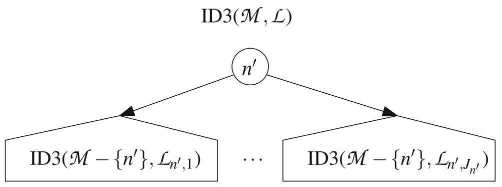
1. If all elements in $\mathcal{L}$ belong to class i return a leaf labelled i.
2. If $\mathcal{M}$ is empty, return a leaf labelled by the most frequent class in $\mathcal{L}$.
3. Else search for the feature n' with the highest IG(R), i.e.,

$$n' = \arg\max_{n \in \mathcal{M}} \text{IG}(\mathcal{L}, n).$$

4. For all $j = 1, \ldots, J_{n'}$ construct a DT by ID3 on the feature set $\mathcal{M} - \{n'\}$ and the training set $\mathcal{L}_{n',j}$. Return a tree with the root labelled by the feature n' whose edges lead to the constructed DTs (cf. Fig. 7.2)

ID3 is a greedy algorithm as in every step a feature is selected by a local optimisation criterion. A global optimum is not guaranteed. Further, ID3 always terminates, as with every recursive call the remaining set of features decreases and the case of an empty feature set is handled separately.

An extension of ID3 are the C4.5 or J48 variants that introduce pruning of sub-trees [2, 4] for increased efficiency. During pruning, a whole sub-tree can be replaced by a leaf, if the error probability is not significantly increased by this substitution. Note



$\mathrm{ID3}(\mathcal{M}, \mathcal{L})$
n'
$\mathrm{ID3}(\mathcal{M} - \{n'\}, \mathcal{L}_{n',1})$
$\cdots$
$\mathrm{ID3}(\mathcal{M} - \{n'\}, \mathcal{L}_{n',J_{n'}})$

Fig. 7.2 Recursive call of the ID3 algorithm for a feature n' that maximises the IG(R) with respect to the classification of $\mathcal{L}$ within $\mathcal{M}$ [1]

that this reduces the number of features, i.e., an inherent feature selection by IG(R) is given. DTs are able to handle missing features both in training and recognition. Further, if both, the feature set and the training set are randomly sub-sampled for construction of an ensemble (cf. Sect. 7.4) of DTs, one speaks of Random Forests (RF), which are known as competitive classifier [5], e.g., to the further introduced classifiers.

7.2.2 Support Vectors

Support Vector Machines (SVM) and Support Vector Regression (SVR) were introduced in [6]. In principle, they base on statistical learning theory, and their theoretic foundation can be interpreted as analogon to electrostatics: Thereby, a training instance corresponds to a charged conductor at a given place in space, the decision function corresponds to an electrostatic potential function and the learning target function to Coulomb's energy [7].

The concept of SVM and SVR unites several theories of machine learning and optimisation: At first, a linear classifier or regressor—similar to a perceptron with linear activation function —is combined with a non-linear mapping into a higher dimensional decision space in order to be able to solve more complex decision tasks. The linear classifier is thereby built based on a subset of the learning instances—the so called 'support vectors'. By that, the danger of overfitting to the learning instances as a whole is limited. The choice of support vectors is achieved by a quadratic optimisation problem.

7.2.2.1 Support Vector Machines

In general, SVM are by that capable to discriminate between two classes, i.e., solve binary decision problems. We will at first focus on this task—the solving of multiple class problems can then be reached by diverse strategies such as one-versus-one pairwise decisions, one-versus-all decisions, or binary-tree-based grouping of decisions.

SVM are trained based on a set $\mathcal{L}$ of L learning instances, where each of the instances is accordingly labelled with its class. For $l = 1, \ldots, L$ the assignment of a pattern instance $\underline{x}_l$ to its class is denoted by $y_l \in \{-1, +1\}$. By definition the patterns $\underline{x}_l$ with $y_l = +1$ are the 'positive' instances, i.e., $\underline{x}_l \in \underline{X}_1$. If $y_l = -1$, $\underline{x}_l$ is a 'negative' instance, i.e., $\underline{x}_l \in \underline{X}_2$. By this we can denote $\mathcal{L}$ as:

$$\mathcal{L} = \{(\underline{x}_l, y_l) \mid l = 1, \ldots, L\} \text{ where } y_l \in \{+1, -1\}. \tag{7.6}$$

The assignment $y_l \in \{-1, +1\}$ simplifies the mathematical handling. In order to be able to strictly separate the according instances in the following, the normal vector $\underline{w}$ and the scalar bias b define the hyper plane $H(\underline{w}, b)$ given as

$$H(\underline{w}, b) = \{\underline{x} \mid \underline{w}^T \underline{x} + b = 0\}. \tag{7.7}$$

The task is now to find the hyper plane in such a way that the conditions

$$\begin{aligned} y_l = +1 &\Rightarrow \underline{w}^T \underline{x}_l + b \geq +1, \\ y_l = -1 &\Rightarrow \underline{w}^T \underline{x}_l + b \leq -1 \end{aligned} \tag{7.8}$$

are fulfilled. Under the condition that a hyper plane exists by which the separation of the (two) classes is possible without misclassification, a normalisation of the side conditions (7.8) can be realised by appropriate scaling of $\underline{w}$ and b [8]. Next, by application of the signed distance $D(\underline{x})$ of a point $\underline{x}$ to the hyper plane H

$$D(\underline{x}) = \frac{\underline{w}^T \underline{x} + b}{||\underline{w}||} \tag{7.9}$$

the margin of separation $\mu_{\mathcal{L}}$ is defined as the minimum of the magnitude of the distances of all points $\underline{x}_1 \ldots \underline{x}_l$ in $\mathcal{L}$ to H:

$$\mu_{\mathcal{L}}(\underline{w}, b) = \min_{l=1,\ldots,L} |D(\underline{x}_l)|. \tag{7.10}$$

In order to reach maximum discriminativity between the two classes, this margin needs to be maximised. To this end, we seek the hyper plane $H^* = H(\underline{w}^*, b^*)$ with maximal value $\mu^*_{\mathcal{L}}(\underline{w}^*, b^*)$ to separate the training instances set $\mathcal{L}$. The according instances $\underline{x}_l^{sv} \in \mathcal{L}$, which satisfy (7.10), are closest to the hyper plane H^* and are called support vectors of H^* with respect to $\mathcal{L}$. Their distance $D^*(\underline{x}_l^{sv})$ to the hyper plane H^* is—owing to the normalisation of the separation condition:

$$D^*(\underline{x}_l^{sv}) = \frac{\pm 1}{||\underline{w}||}. \tag{7.11}$$

As a consequence, a corridor between the positive and negative instances results of the width $2\,||\underline{w}||^{-1}$. Its border is given by the support vectors which are shown in Fig. 7.3.

Instead of the maximisation of the width of the corridor one can minimise the expression $\frac{1}{2}\underline{w}^T \underline{w}$. The resulting funtion to be minimised is strictly convex and

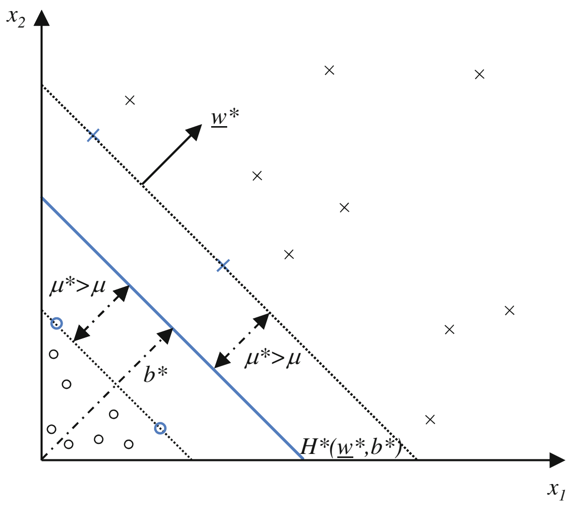


Fig. 7.3 Example of an optimal hyper plane $H^*(\underline{w}, b)$ (*lighter shaded*) in two dimensional space with maximum margin of separation μ^* (*dashed parallel lines*). "x" and "o" indicate exemplary instances of the two classes to be separated

posseses a unique minimum $\underline{w}^*$. From (7.8) result linear side conditions for the optimisation:

$$y_l\,(\underline{w}^T\underline{x}_l + b) - 1 \geq 0 \text{ with } l = 1, \ldots, L. \tag{7.12}$$

To solve this boundary value problem one can use Langrange multipliers. In [6] this is explained in detail.

In the general, non-trivial case, there does not exist—as opposed to the previously made assumption—a hyper plane to separate a training instances set $\mathcal{L}$ flawlessly. In this case the equations in (7.8) are extended by so called slack variables $\xi_l \geq 0, l = 1, \ldots, L$. This allows to stay with the approach, as vectors which cross the hyper plane may be placed on the 'wrong side':

$$\begin{aligned} y_l = +1 &\Rightarrow \underline{w}^T\underline{x}_l + b \geq +1 - \xi_l, \\ y_l = -1 &\Rightarrow \underline{w}^T\underline{x}_l + b \leq -1 + \xi_l. \end{aligned} \tag{7.13}$$

By that, the expression

$$\frac{1}{2}\underline{w}^T\underline{w} + G \cdot \sum_{l=1}^{L}\xi_l \tag{7.14}$$

needs to be minimised, where G is a free error weighting factor that needs to be determined. It can be shown that this optimisation—also called a 'primal problem'—is equivalent to a 'dual problem' of the maximisation of

$$\sum_{l=1}^{L}a_l - \frac{1}{2}\sum_{k=1}^{L}\sum_{l=1}^{L}a_k\,a_l\,y_k\,y_l(\underline{x}_k^T\underline{x}_l), \tag{7.15}$$

with the side conditions

$$0 \leq a_l \leq C, l = 1, \ldots, L, \tag{7.16}$$

$$\sum_{l=1}^{L} a_l \, y_l = 0. \tag{7.17}$$

The hyper plane is then defined by

$$\underline{w} = \sum_{l=1}^{L} a_l \, y_l \, \underline{x}_l, \tag{7.18}$$

$$b = y_{l^*}(1 - \xi_{l^*}) - \underline{x}_{l^*}^T \underline{w}_{l^*}. \tag{7.19}$$

Thereby l^* represents the index of the vector $\underline{x}_l$ with the largest coefficient a_l. The normal vector $\underline{w}$ is thus represented as weighted sum of training instances with the coefficients $a_l \leq C, l = 1, \ldots, L$, where C is another free parameter to be determined. By the introduction of the weighting coefficients the slack variables ξ_l disappear in the optimisation problem. The support vectors are then the training instances $\underline{x}_l$ that satisfy $a_l > 0$.

By this, L^2 terms of the form $\underline{x}_k^T \underline{x}_l$ result, which can be summarised as a matrix. One of the frequently used and highly efficient methods for the recursive computation of this matrix and by that solving of the dual problem is the Sequential Minimal Optimisation (SMO), which is introduced in detail in [9]. The classification by SVM is now given by the function $d_{\underline{w},b} : \underline{X} \rightarrow \{-1, +1\}$,

$$d_{\underline{w},b}(\underline{x}) = \text{sgn}(\underline{w}^T \underline{x} + b) \tag{7.20}$$

where

$$\text{sgn}(u) = \begin{cases} 1 & u \geq 0 \\ -1 & u < 0. \end{cases} \tag{7.21}$$

So far, we are only able to solve pattern recognition problems that assign the instances belonging to the (two) different classes with a certain acceptable error by a hyper plane in the space $\underline{X}$. This is referred to as linear seperation problem. Aiming at classes that can only be separated non-linearly, one applies the so called ‘kernel trick’ [10]. Figure 7.4 depicts an exemplary two-class problem in one-dimensional space, which can only be solved linearly by a mapping into a higher (two-)dimensional space—without error in the given example.

In general, such a transformation is given by the mapping

$$\Phi : \underline{X} \rightarrow \underline{X}^{'}, \quad \dim(\underline{X}^{'}) > \dim(\underline{X}). \tag{7.22}$$

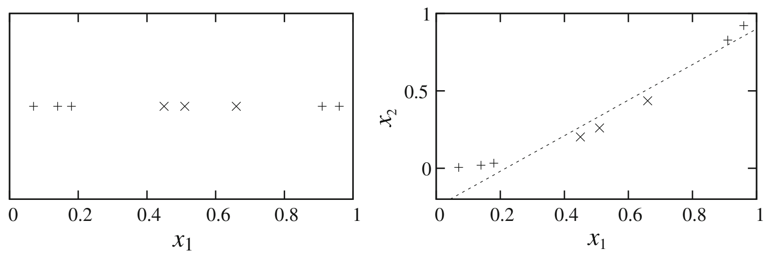


Fig. 7.4 Solving of an exemplary two-class problem by mapping into higher dimensional space: While in the one-dimensional (original) space the problem cannot be solved linearly, mapping by the function $\Phi : x_1 \mapsto (x_1, x_1^2)$ allows for error-free separation in the new two-dimensional space [1]

The normal vector $\underline{w}$ then results in

$$\underline{w} = \sum_{l:a_l>0} a_l \, y_l \, \Phi(\underline{x}_l). \tag{7.23}$$

The decision function $d_{\underline{w},b}(\underline{x})$ results—applying Φ—in:

$$d_{\underline{w},b}(\underline{x}) = \mathrm{sgn}(\underline{w}^T \Phi(\underline{x}) + b). \tag{7.24}$$

As

$$\underline{w}^T \Phi(\underline{x}) = \sum_{l:a_l>0} a_l \, y_l \, \Phi(\underline{x}_l)^T \Phi(\underline{x}), \tag{7.25}$$

the transformation Φ is explicitly neither needed for the estimation of the parameters of the classifier, nor for the classification. Instead a so called 'kernel function' $K^{\Phi}(\underline{x}, \underline{x}')$ is being defined, with the condition

$$K^{\Phi}(\underline{x}, \underline{x}') = \Phi(\underline{x})^T \Phi(\underline{x}'). \tag{7.26}$$

The kernel function additionally needs to be positively semi-definite, symmetric, and fulfil the Cauchy-Schwarz inequality. The optimal kernel function for a given classification or regression problem can only be found empirically. However, recently so called multi kernels try to overcome the search for optimal kernel functions [11]. Most frequently used kernel functions comprise:

- Polynomial kernel:

$$K_p^{\Phi}(\underline{x}, \underline{x}') = (\underline{x}^T \underline{x}' + 1)^p, \tag{7.27}$$

  where p is the polynomial order,

- Gaussian kernel (radial basis function, RBF):

$$K_{\sigma}^{\Phi}(\underline{x}, \underline{x}') = e^{\frac{||\underline{x}-\underline{x}'||^2}{2\sigma^2}}, \tag{7.28}$$

  where σ is the standard deviation of the Gaussian, and
- Sigmoid kernel:

$$K_{k,\Theta}^{\Phi}(\underline{x}, \underline{x}') = \tanh(k(\underline{x}^T \underline{x}') + \Theta), \tag{7.29}$$

  where k is the amplification, and Θ the off-set.

The application of the kernel function K^{Φ} instead of the transformation Φ considerably reduces the required computation effort and allows for practical application of SVM and SVR when coping with high dimensional problems, as can be seen by the example of the polynomial kernel: to compute a polynomial of the order p in the space $\underline{X}$,

$$\binom{\dim(\underline{X}) + p}{p} \approx \frac{\dim(\underline{X})^p}{p!} \tag{7.30}$$

terms would need to be calculated, while the computation employing the polynomial kernel independently of p requires only approximately $\dim(\underline{X})$ operations. There exist manifold further kernels for special requirements, such as the KL divergence kernel frequently used in Gaussian Mixture Model (GMM)-SVM 'super vector' construction.

The last kernel that is introduced is a special solution for symbolic, i.e., non-numeric input: The recent string subsequence kernel (SSK) approach [12] makes use of a mapping from text information to a high dimensional feature space without explicit calculation of features. Based on the theory of Support Vector Machines, the idea of kernel mapping is extended for strings as input parameters. Thus, a special kernel for text information is provided. The idea behind is to observe small substrings in a given string. For a predefined substring length, all possible substrings form a feature space in which a string can be represented. The numeric value of each feature depends on the substring occurrence frequency in the string and on the degree of contiguity. For example, the substring *"ser"* exists in the word "$\underline{\textit{ser}}\textit{ene}$" as well as in "$\underline{\textit{s}}\textit{u}\underline{\textit{p}}\textit{e}\underline{\textit{r}}\textit{b}$", but with a different degree of contiguity. This degree is weighted by a decay factor $\lambda \in [0, 1]$ which penalises non-contiguous substrings. Taking non-continuous substrings into account is a specific characteristic of the string kernel method.

The transformation of a string s into the feature space is done by a mapping $\Phi(s)$ which can be calculated numerically as described in [12]. Analog to the Support Vector Machines' theory, this mapping does not have to be done explicitly. An implicit calculation is done by using a kernel function:

$$K^{\Phi}(s, t) = \langle \Phi(s), \Phi(t) \rangle . \tag{7.31}$$

This kernel function is part of the decision function for SVM or SVR. The inner product calculated by the kernel can be seen as a numeric measure of similarity between two strings s and t. The calculation of this string subsequence kernel can further be simplified due to recursive computation [12], making the procedure practicable.

7.2.2.2 Support Vector Regression

Let us now have a very short introduction to SVR. Again, we first consider a set of training patterns $\mathcal{L}$, but now with numeric values $y_l \in \mathbb{R}$. The goal of SVR is to find a regression function $f(\underline{x})$ that has at the most a deviation of ϵ from the actually obtained targets and, at the same time, is as flat as possible. For a linear regression function,

$$f(\underline{x}) = \underline{w}^T \underline{x} + b \tag{7.32}$$

described by a vector $\underline{w}$ and a scalar b, this flatness can be achieved by minimising the dot product $\underline{w}^T \underline{w}$ under the conditions:

$$\begin{aligned} y_l - \underline{w}^T \underline{x}_l - b &\leq \epsilon, \\ \underline{w}^T \underline{x}_l + b - y_l &\leq \epsilon. \end{aligned} \tag{7.33}$$

Because there are only few cases where all y_l can be linearly estimated within a range between $\pm\epsilon$, non-negative slack variables ξ_l and ξ_l^* are introduced in analogy to SVM, allowing vectors to lie outside this range of $\pm\epsilon$:

$$\begin{aligned} y_l - \underline{w}^T \underline{x}_l - b &\leq \epsilon + \xi_l, \\ \underline{w}^T \underline{x}_l + b - y_l &\leq \epsilon + \xi_l^*. \end{aligned} \tag{7.34}$$

As in the case of SVM, the optimisation is done with Lagrangian multipliers, leading to:

$$\begin{aligned} &\frac{1}{2} \underline{w}^T \underline{w} + C \sum_{l=1}^{L} (\xi_l + \xi_l^*) - \sum_{l=1}^{L} (\eta_l \xi_l + \eta_l^* \xi_l^*) \\ &- \sum_{l=1}^{L} \alpha_l (\epsilon + \xi_l - y_l + \underline{w}^T \underline{x}_l + b) \\ &- \sum_{l=1}^{L} \alpha_l^* (\epsilon + \xi_l^* + y_l - \underline{w}^T \underline{x}_l + b). \end{aligned} \tag{7.35}$$

The optimisation problem is to minimise the Lagrangian multiplier with respect to $\underline{w}$, b and the Lagrangian multipliers $\alpha_l, \alpha_l^*, \eta_l, \eta_l^*$ $(l = 1, \ldots, L)$. The complexity parameter $C > 0$ determines the penalty for regression errors larger than ϵ.

As a further analogy to SVM, the solution of the optimisation problem shows that the vector $\underline{w}_o$ for the regression function searched for can be written as a linear combination of vectors in the test set [13]:

$$\underline{w}_o = \sum_{l=1}^{L} (\alpha_l - \alpha_l^*) \underline{x}_l, \tag{7.36}$$

and thus, the linear regression function becomes

$$f(\underline{x}) = \sum_{l=1}^{L} (\alpha_l - \alpha_l^*) \underline{x}_l^T \underline{x} + b. \tag{7.37}$$

In consequently analogous manner to SVM theory, the algorithm for SVR is described by dot products between training vectors $\underline{x}_l$ and the new, unseen pattern vector $\underline{x}$, whereas only those training vectors are relevant for which the Lagrangian multipliers $(\alpha_l - \alpha_l^*) \neq 0$. These are the *support vectors* for SVR. Geometrically interpreted these are the training vectors which have an absolute estimation error of exactly ϵ.

As in the SVM case, the model is extended to solve non-linear regression tasks. This is done by applying the same kernel trick. The kernel function can be built into the regression function in Eq. (7.37), where it substitutes the dot product $\underline{x}_l^T \underline{x}$:

$$f(\underline{x}) = \sum_{l=1}^{L} (\alpha_l - \alpha_l^*) K^{\Phi}(\underline{x}_l, \underline{x}) + b. \tag{7.38}$$

The function in this form makes SVR an efficient algorithm for regression tasks.

7.2.3 Neural Networks

This section gives a short introduction to Artificial Neural Networks (ANN) with a focus on (bidirectional) Long-Short-Term Memory (BLSTM) networks.

ANNs are capable of learning practically arbitrary functions [14], and belong to the most popular learning algorithms, since McCulloch's and Pitts's first mathematical models in the year 1943 [15] that still provide the basis for today's ANN [16]. Their inspiration is given by neural networks in the central nervous system of vertebrates. The central information processing unit thereby is the neuron. Via its axon the neuron emits a certain activity by electrical pulses [17]. These impulses are propagated to the synaptic connection of other neurons via a branched network. The activity of a neuron is based on its cumulative input activation. In the nature, a higher activity results in a higher impulse frequency. Decisive is in general a threshold—usually approximated by a non-linear transfer function. Overall, a neural network consists of neurons and

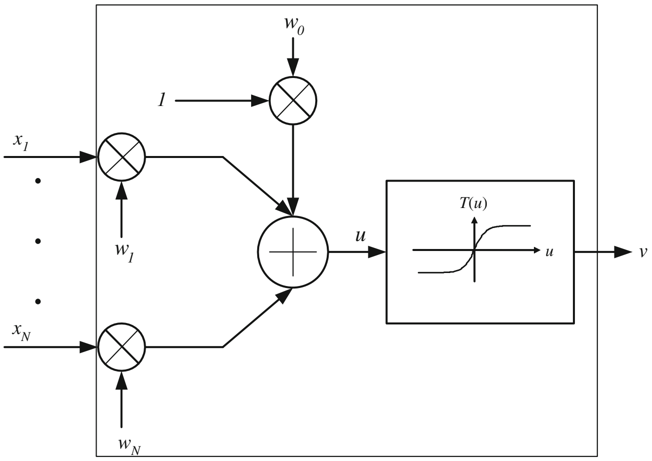


Fig. 7.5 Exemplary neuron

their directed connections. It is fully described by the network topology and weights, the computation type of its units and the encoding of the output. Figure 7.7 shows an example of an ANN. At the N input neurons the values of the feature vector $\underline{x} = \{x_i\}$ with $i = 1, \ldots, N$ are input. These values are weighted by the weights w_i with $i = 0, \ldots, N$ that can be written as $\underline{w} = \{w_i\}$. The weight w_0 is the 'bias'—a permanent additive offset. In the next part, a summation of the weighted inputs takes place. Its result u is then input into the—as stated usually non-linear—transfer function $T(u)$. The output of this function is v at the output of the neuron. In most cases, one aims at a steep decision function. An according visualisation of a neuron is given in Fig. 7.5.

Popular transfer functions are in particular the sigmoid function (cf. Fig. 7.6)

$$T(u) = \frac{1}{1 + e^{-\alpha u}}, \tag{7.39}$$

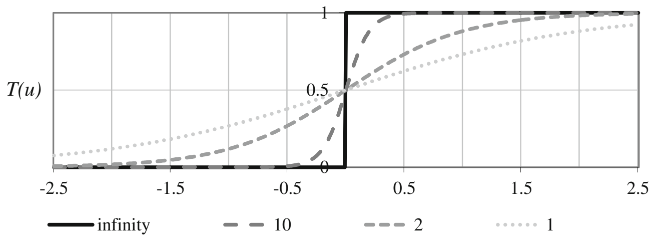


Fig. 7.6 Sigmoid function with different values for the steepness parameter α. In the case $\alpha \to \infty$ the function approximates a threshold decision

where α is the steepness parameter, the hyperbolic tangent function as special case of the sigmoid function with additive offset, and the unit step

$$T(u) = \begin{cases} 0 & \text{if } u < 0 \\ 1 & \text{if } u \geq 0 \end{cases} . \tag{7.40}$$

The sigmoid function is particularly popular owing to its approximation of an ideal threshold decision (cf. Fig. 7.6) while being differentiable. The latter will be needed throughout the training of the network.

A multiplicity of different network topologies exist, of which the most important will be introduced next.

7.2.3.1 Feed Forward Neural Networks

The most commonly used form of feed forward neural networks (FNN) is the multilayer perceptron (MLP) [18]: It consists of a minimum of three layers, one input layer—typically without processing—, one or more hidden layers, and an output layer. All connections feed forward from one layer to the next without any backward connections. MLPs classify all input feature vectors over time independently. In general, encoding of the outputs $\hat{y}_j$ with $j = 1, \ldots, M$ of the last layer that can be written as vector $\underline{\hat{y}}$ is required. A popular way is to provide one output neuron for regression and one per class in the case of classification. As an advantage, this provides a measure of confidence of the network: The 'softmax' function as a transfer function normalises the sum of all outputs to one in order to allow for interpretation as posterior probability $P(j|\underline{x})$ of the final output:

$$P(j|\underline{x}) = \hat{y}_j = \frac{e^u}{\sum_{j=1}^{M} e^u} . \tag{7.41}$$

In the recognition phase the computation is processed step-wisely from the input layer to the output layer. Per layer the weighted sum of the inputs from the previous layer is computed for each neuron and weighted by the non-linearity. Using the softmax function at the outputs, and the named encoding, the recognised class is assigned by maximum search. As an alternative, one can choose, e.g., a binary encoding of the classes with the network's outputs.

7.2.3.2 Back Propagation

Among the multiplicity of learning algorithms for ANNs, the gradient descent-based back propagation algorithm [19] is among the most popular ones and allowed for the break-through of ANN. Let $\underline{W} = \{\underline{w}_j\}$ summarise the weight vectors $\underline{w}_j$ of a layer with $j = 1, \ldots, J$ and J being the number of neurons in this layer. As target function to measure the progress of (supervised) learning, the MSE $E(\underline{x}, \underline{W})$ between the gold standard y and the network output $\hat{y} = f(\underline{x}, \underline{W})$ is used—for simplification we consider the case of a single output as in regression—an extension to multiple outputs is straight forward:

$$E(\underline{x}, \underline{W}) = |y - \hat{y}|^2 \tag{7.42}$$

Other target functions are frequently used, such as McClelland error or cross-entropy. After an initialisation of weights, e.g., by random, three steps follow for the back propagation:

1. Forward pass as 'normal' pass as in the recognition phase.
2. Computation of the MSE according to Eq. (7.42).
3. Backward pass with weight adaptation by the corrective term:

$$w_i \rightarrow w_i + \Delta w_i = w_i - \beta \cdot \frac{\delta E(\underline{x}, \underline{W})}{\delta w_i}, \tag{7.43}$$

   where β is the step size, which is to be determined empirically, and w_i is an individual weight within a neuron.

As a stopping criterion of the iterative updating of the weights one can either use a maximum number of iterations or a minimal change of the error [20]. A 'good' parameter set can only be determined empirically and based on experience. However, approaches exist to learn these. To avoid over fitting, a sufficient number of training instances is required as compared to the number of parameters in the network and the dimensionality of the feature vector. An alternative is resilient propagation that incorporates the last change of weights into the current change of weights [21]. By learning the weights, ANNs are able to cope with redundant feature information. The learning process is further discriminative as the information over all classes is learnt at a time [17]. Their highly parallel processing is one of the main advantages for efficient implementation. If the temporal context of a feature vector is relevant, this context must be explicitly fed to the network, e.g., by using a fixed width sliding window that combines several feature vectors to a 'super vector', as in [22].

7.2.3.3 Recurrent Neural Networks

Another technique for introducing past context to neural networks is to add backward (cyclic) connections to FNNs. The resulting network is called a recurrent neural network (RNN). RNNs can theoretically map from the entire history of previous inputs to each output. The recurrent connections implicitly form a kind of memory, which allows input values to persist in the hidden layer(s) and influence the network output in the future. RNNs can be trained by back propagation through time (BPTT) [23]. In BPTT, the network is first unfolded over time. The training then is similar as if training a FNN with back propagation. However, each epoch must run through the output observations in sequential order. Details are found in [23]. If in a RNN future

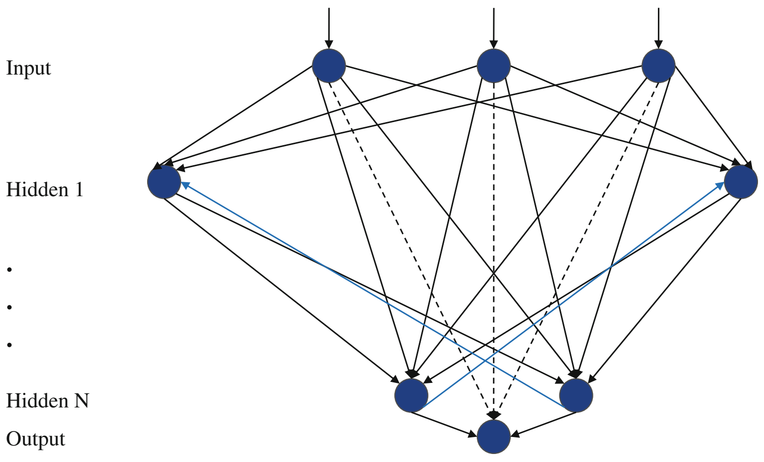


Fig. 7.7 A RNN with two hidden layers and a single output neuron for regression or binary classification. Dashed connection are an example of an architectural variation. The blue connections are examples of recurrent connections. Other popular ways of recurrent connections include such from the output nodes of a layer to its own input nodes

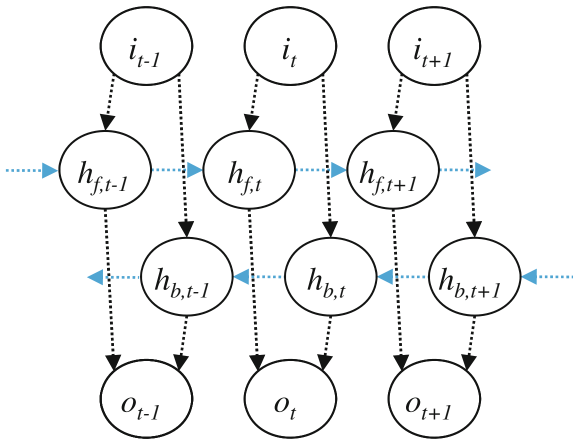


Fig. 7.8 Structure of a bidirectional network with input i, output o, as well as two hidden layers that processes the input sequence forwards (h_f) and backwards (h_b) over time t

context is also required, a delay between the input values and the output targets can be introduced. An example is shown in Fig. 7.7.

A more elegant incorporation of future temporal context is provided by a bidirectional recurrent neural network (BRNN). Two (sets of) separate hidden layers are used instead of one, both connected to the same input and output layers. The first processes the input sequence forwards and the second backwards. The network therefore has always access to the complete past and the future temporal context in a symmetrical way, without bloating the input layer size or displacing the input values from the corresponding output targets. Figure 7.8 visualises this principle.

However, they must have the complete input sequence at hand before it can be processed.

7.2.3.4 Long Short-Term Memory

Although BRNNs have access to both past and future information, the range of temporal context is limited to a few frames due to the ‘vanishing gradient’ problem [24]. The influence of an input value decays or blows up exponentially over time, as it cycles through the network with its recurrent connections and gets dominated by new input values. To overcome this deficiency, a method called Long Short-Term Memory (LSTM) was introduced in [25]. In a LSTM hidden layer, the non-linear units are replaced by LSTM memory blocks (cf. Fig. 7.10). Each block contains one or more self connected linear memory cells. By that, they are able to overcome the vanishing gradient problem and can learn the optimal amount of contextual information relevant for the learning task. Figure 7.9 depicts this vanishing gradient problem for RNN and how it is overcome by LSTM (right).

A LSTM layer is composed of recurrently connected memory blocks, each of which contains one or more memory cells, along with three multiplicative ‘gate’ units: the input, output, and forget gates. The gates perform functions analogous to read, write, and reset operations. More specifically, the cell input is multiplied by the activation of the input gate, the cell output by that of the output gate, and the previous cell values by the forget gate (cf. Fig. 7.10). Usually, one can employ the same non-linear transfer function for these gates, denoted as T_g in the ongoing. A popular choice is a hyperbolic tangent function. The transfer function of the ‘original’ neuron (top neuron Fig. 7.10) is often chosen as a sigmoid function and referred to by T_i in the ongoing, as it functions as the actual input neuron of a LSTM cell. The output transfer function of the LSTM cell after the ‘error carousel’ (EC) is denoted as T_o from now on. Sigmoid or softmax functions are popular choices for this function. The outgoing weight of the EC is chosen as 1 to realise the storage effect by an auto-transition of one.

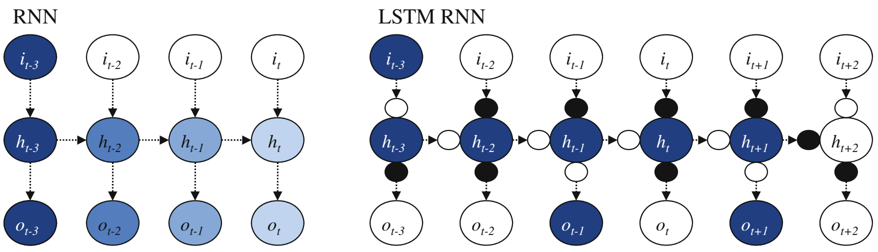


Fig. 7.9 Vanishing gradient problem of a RNN *(left)* and overcoming it by use of LSTM *(right)*. Lighter shading indicates decreased memory of past events. i_t, h_t, o_t represent the input, hidden, and output layers at time t, respectively

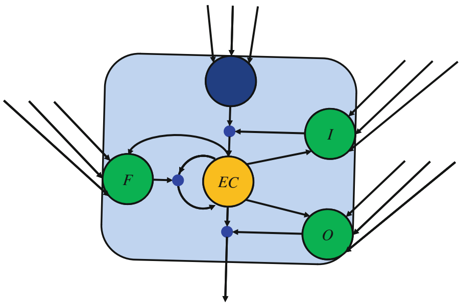


Fig. 7.10 LSTM memory block consisting of one memory cell: input, output, and forget gate collect activations from inside and outside the block which control the cell through multiplicative units (depicted as small circles); input, output, and forget gate scale input, output, and internal state respectively; a recurrent connection of fixed weight 1.0 maintains the internal state

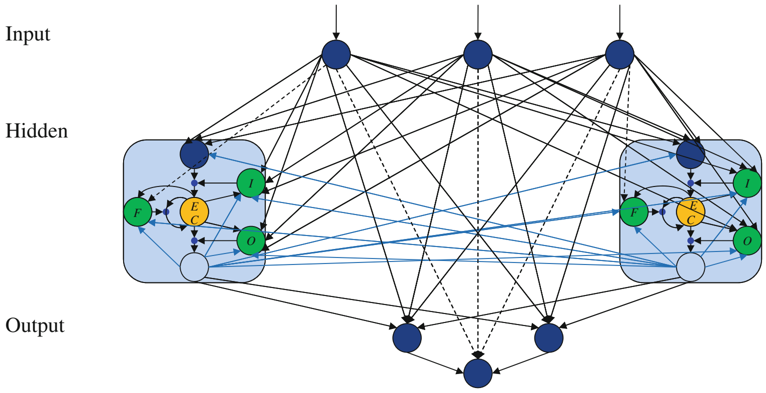


Fig. 7.11 An exemplary layout of a RNN with LSTM cells

The overall effect is to allow the network to store and retrieve information over long periods of time. For example, as long as the input gate remains closed, the activation of the cell will not be overwritten by new inputs and can therefore be made available to the net much later in the sequence by opening the output gate.

Figure 7.11 depicts LSTM cells' exemplary integration in a RNN. If $\alpha_{\text{in},t}$ denotes the activation of the input gate at time t *before* the activation function T_g has been applied and $\beta_{\text{in},t}$ represents the activation *after* application of the activation function, the input gate activations (forward pass) can be written as

$$\alpha_{\text{in},t} = \sum_{i=1}^{I} w_{i,\text{in}} x_{i,t} + \sum_{h=1}^{H} w_{h,\text{in}} \beta_{h,t-1} + \sum_{c=1}^{C} w_{c,\text{in}} s_{c,t-1} \tag{7.44}$$

and

$$\beta_{\text{in},t} = T_g(\alpha_{\text{in},t}), \tag{7.45}$$

respectively. The variable $w_{i,j}$ corresponds to the weight of the connection from unit i to unit j while 'in', 'for', and 'out' refer to input gate, forget gate, and output gate, respectively (cf. Eqs. 7.46 and 7.50). Indices i, h, and c count the inputs $x_{i,t}$, the cell outputs from other blocks in the hidden layer, and the memory cells, while I, H, and C are the number of inputs, the number of cells in the hidden layer, and the number of memory cells in one block. Finally, $s_{c,t}$ corresponds to the *state* of a cell c at time t, meaning the activation of the linear cell unit.

Similarly, the activation of the forget gates before and after applying T_g can be calculated as follows:

$$\alpha_{\text{for},t} = \sum_{i=1}^{I} w_{i,\text{for}} x_{i,t} + \sum_{h=1}^{H} w_{h,\text{for}} \beta_{h,t-1} + \sum_{c=1}^{C} w_{c,\text{for}} s_{c,t-1} \tag{7.46}$$

$$\beta_{\text{for},t} = T_g(\alpha_{\text{for},t}). \tag{7.47}$$

The memory cell value $\alpha_{\text{c},t}$ is a weighted sum of inputs at time t and hidden unit activations at time $t-1$:

$$\alpha_{\text{c},t} = \sum_{i=1}^{I} w_{i,\text{c}} x_{i,t} + \sum_{h=1}^{H} w_{h,\text{c}} \beta_{h,t-1}. \tag{7.48}$$

To determine the current state of a cell c, the previous state is scaled by the activation of the forget gate and the input $T_i(\alpha_{\text{c},t})$ by the activation of the input gate:

$$s_{\text{c},t} = \beta_{\text{for},t} s_{\text{c},t-1} + \beta_{\text{in},t} T_i(\alpha_{\text{c},t}). \tag{7.49}$$

The computation of the output gate activations follows the same principle as the calculation of the input and forget gate activations, however, this time the *current* state $s_{c,t}$ is considered, rather than the state from the previous time step:

$$\alpha_{\text{out},t} = \sum_{i=1}^{I} w_{i,\text{out}} x_{i,t} + \sum_{h=1}^{H} w_{h,\text{out}} \beta_{h,t-1} + \sum_{c=1}^{C} w_{c,\text{out}} s_{c,t} \tag{7.50}$$

$$\beta_{\text{out},t} = T_g(\alpha_{\text{out},t}). \tag{7.51}$$

Finally, the memory cell output is determined as

$$\beta_{\text{c},t} = \beta_{\text{out},t} T_o(s_{\text{c},t}). \tag{7.52}$$

Note that the initial version of the LSTM architecture contained only input and output gates. Forget gates were added later [26] in order to allow the memory cells to reset themselves whenever the network needs to *forget* past inputs.

LSTM networks can be trained by BPTT. They have shown remarkable performance in a variety of pattern recognition tasks, including phoneme classification [27], handwriting recognition [28], keyword spotting [29], affective computing [30], and driver distraction detection [31]. Combining bidirectional networks with LSTM leads to bidirectional LSTM (BLSTM). Further details on the LSTM technique can be found in [28].

7.3 Dynamic Learning Algorithms

Audio is sequential, and an endpointed audio stream $\underline{X} = \{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_T\}$ accordingly yields a series of T feature vectors. So far, however, we mostly dealt with classification of single feature vectors without use of temporal context. One exception were the different types of RNN that modelled such context, as discussed above. But even these are not able to 'warp' in time, i.e., to handle different tempo deviations between, e.g., two musical pieces or stretching or shortening, e.g., of vowels while speaking. The most frequently encountered algorithm for audio sequence classification are HMMs [32] as a simple form of DBNs. This property is owed to their ability of dynamic modelling throughout different hierarchy levels and a well-formulated stochastic framework. In ASR, for example, the extracted feature stream is first modelled on the phoneme level. On a higher level, these phonemes are then used to form words. Each class i is modelled by a HMM that represents the probability $P(\underline{X}|i)$, where $\underline{X}$ is called the 'observation', which is generated by the HMM.

A Markov model can be seen as finite state automaton that may change its state at any step in time. In a HMM, at each step in time t a feature vector $\underline{x}_t$ is being generated depending on the current state s and the emission probability $b_s(\underline{x})$. The probability of a transition from state j to state k is expressed by the state transition probability $a_{j,k}$ [33]. The probabilities $a_{0,j}$ are needed to enter the model in a state j with a certain probability. In order to simplify calculation, a non-emitting initial state s_0 and a non-emitting final state s_F can be defined [1]. In Fig. 7.12 the structure of such a model is depicted. In the example, the most frequently used type of HMM for audio processing is depicted—the so-called left-right model. In this model type, the state number cannot decrease over time. In the 'linear' model, no state can be skipped. Other topologies allow for a state skip, such as the Bakis model in which one state may be skipped. If any state can be reached from any other state with a probability above zero, the topology is referred to as 'ergodic'.

One speaks of a 'hidden' Markov model, as the sequence of states remains unknown—only the observations sequence is known [32]. Note the 'Markov property' that the conditional probability distribution of the hidden variable $s(t)$ at time step t, given the values of the hidden variable s at all times, depends only on the hidden variable $s(t-1)$, i.e., values at earlier steps in time have no influence [34].

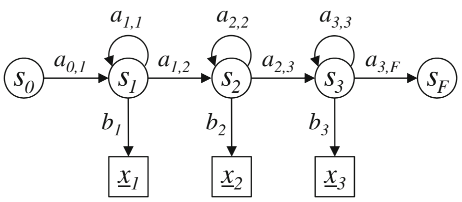


Fig. 7.12 Example of an instantiated linear left-right HMM with three emitting states. Squares indicate observations, circles represent switching states, arrows denote conditional dependencies

Further, the observation $\underline{x}(t)$ only depends on the value of the current state's hidden variable $s(t)$.[1]

The needed probability $P(\underline{X}|i)$ can be computed by summation over all possible state sequences:

$$P(\underline{X}|i) = \sum_{Seq} a_{s_0,s_1} \prod_{t=1}^{T} b_{s_t}(\underline{x}_t) a_{s_t,s_{t+1}} , \tag{7.53}$$

where Seq stands for the set of all possible state sequences. For the efficient computation of this probability, the forward algorithm is typically applied as is introduced in Sect. 7.3.1. Instead of a summation over all state sequences, the Viterbi algorithm considers only the most probable state sequence, which results in a speed-up at the cost of the global optimum [34]:

$$\hat{P}(\underline{X}|i) = \max_{Seq} \left\{ a_{s_0,s_1} \prod_{t=1}^{T} b_{s_t}(\underline{x}_t) a_{s_t,s_{t+1}} \right\}. \tag{7.54}$$

In the recognition phase the class i is decided for according to the model that is assigned the highest probability $P(\underline{X}|i)$. This requires the parameters $a_{j,k}$ and $b_s(\underline{x}_t)$ to be known for each model. Just as for the previous static classifiers, these are determined in a training phase given a large set of training instances. The popular method to this end is the forward-backward algorithm which is also described in Sect. 7.3.1.

In most Intelligent Audio Analysis application scenarios the emission probabilities $b_s(\underline{x}_t)$ are modelled by Gaussian mixtures. Such mixtures are linear superpositions of Gaussian functions. With the number of mixture components M and the 'mixture weight' of the m-th component $c_{s,m}$ the emission probability density function (PDF) can be determined as [34]:

$$b_s(\underline{x}_t) = \sum_{m=1}^{M} c_{s,m} \mathcal{N}(\underline{x}_t; \underline{\mu}_{s,m}, \underline{\Sigma}_{s,m}), \tag{7.55}$$

[1] Note that for better readability, the time t is in this section used in the subscript or argument following [32].

where $\mathcal{N}(\cdot; \underline{\mu}, \underline{\Sigma})$ is a multivariate Gaussian density with mean vector $\underline{\mu}$ and the covariance matrix $\underline{\Sigma}$. Apart from such 'continuous' HMMs, also 'discrete' HMMs are used. These use conditional probability tables for discrete observations $b_s(\underline{x}_t)$.

7.3.1 Estimation

The parameters of HMMs can be determined by the Baum-Welch estimation [35]—a case of generalised Expectation Maximisation (EM). If the ML estimates of the means and covariances per state s are to be computed, one has to take into account that each observation vector $\underline{x}$ contributes to the parameters of a state. This comes, as the overall probability of an observation bases on the summation of all possible state sequences. Thus, the Baum-Welch estimation assigns each observation to each state in proportion to the state probability at the observation of the respective feature vectors. With $L_{s,t}$ as the probability to be in state s at time step t, the Baum-Welch estimation for the means and covariances of a single Gaussian PDF is obtained as (the hat symbol marks estimated parameters in the following equations):

$$\hat{\underline{\mu}}_s = \frac{\sum_{t=1}^{T} L_{s,t}\underline{x}_t}{\sum_{t=1}^{T} L_{s,t}} \tag{7.56}$$

$$\hat{\underline{\Sigma}}_s = \frac{\sum_{t=1}^{T} L_{s,t}(\underline{x}_t - \underline{\mu}_s)(\underline{x}_t - \underline{\mu}_s)^T}{\sum_{t=1}^{T} L_{s,t}}. \tag{7.57}$$

The 'up-mixing' to several mixture components is reached in a simple way by considering the mixture components as sub-states. In these sub-states, the state transition probabilities correspond to the mixture weights. The state transition probabilities are estimated by the relative frequencies

$$\hat{a}_{j,k} = \frac{A_{j,k}}{\sum_{s=1}^{S} A_{j,s}}, \tag{7.58}$$

where $A_{j,k}$ denotes the number of transitions from state j to state k, and S denotes the number of states of the HMM.

For the computation of the probability $L_{s,t}$ the forward-backward algorithm is applied. The 'partial' forward probability $\alpha_s(t)$ for a HMM that represents the class i is defined as:

$$\alpha_s(t) = P(\underline{x}_1, \ldots, \underline{x}_t, s_t = s|i). \tag{7.59}$$

This can be interpreted as the joint probability of the observation of the first t feature vectors and being in state s at time step t. The following recursion allows for an efficient computation of the forward probability, where S is the number of emitting states:

$$\alpha_s(t) = \left[\sum_{j=1}^{S} \alpha_j(t-1) a_{j,s} \right] b_s(\underline{x}_t) \tag{7.60}$$

The according backward probability represents the joint probability of the observation from time step $t + 1$ to T:

$$\beta_s(t) = P(\underline{x}_{t+1}, \ldots, \underline{x}_T | s_t = s, i). \tag{7.61}$$

It can be determined by the recursion:

$$\beta_j(t) = \sum_{s=1}^{S} a_{j,s} b_s(\underline{x}_{t+1}) \beta_s(t+1). \tag{7.62}$$

To compute the probability to be in a state at a given time step, one has to multiply the forward and backward probabilities:

$$P(\underline{X}, s_t = s | i) = \alpha_s(t) \cdot \beta_s(t). \tag{7.63}$$

By that, L_{st} can be determined by:

$$L_{st} = P(s_t = s | \underline{X}, i) = \frac{P(\underline{X}, s_t = s | i)}{p(\underline{X} | i)} = \frac{1}{p(\underline{X} | i)} \cdot \alpha_s(t) \cdot \beta_s(t). \tag{7.64}$$

Assuming the last state S at the moment in time of the last observation $\underline{x}_T$ needs to be taken, the probability $P(\underline{X} | M_t)$ equals $\alpha_S(T)$. By that, the Baum-Welch estimation can be executed as described.

The Viterbi algorithm is usually applied in the recognition phase. It is similar to the forward probability. However, the summation is replaced by a maximum search to allow for the following forward recursion:

$$\phi_s(t) = \max_j \{\phi_j(t-1) a_{j,s}\} b_s(\underline{x}_t), \tag{7.65}$$

where $\phi_s(t)$ is the ML probability of the observation of the vectors $\underline{x}_1$ to $\underline{x}_t$ and being in state s at time step t for a given HMM representing class i. Thus, the estimated ML probability $\hat{P}(\underline{X} | i)$ equals $\phi_S(T)$.

7.3.2 Hierarchical Decoding

HMM are in particular suited for decoding, i.e., segmenting and recognising continuous audio streams. In addition, their probabilistic formulation allows for elegant hierarchical analysis in order to unite knowledge at different levels as stated. Typical tasks include continuous speech recognition or chord labelling in music. Let $\mathcal{S}$ be a 'sequence' such as a spoken sentence or musical phrase. Then, the sequence $\underline{X}$ of T feature vectors stems from the phrase $\mathcal{S}$ [36]. The classifier now provides an estimate $\hat{\mathcal{S}}$ for the sequence aiming at the best match with the actual sequence $\mathcal{S}$. According to Bayes' decision rule a decision is optimal if the classifier picks the class which—based on the current observation—has the highest probability. For the optimal decision it thus needs to hold:

$$p(\hat{\mathcal{S}}|\underline{X}) = \max_{\mathcal{S}_j} \; p(\mathcal{S}_j|\underline{X}), \tag{7.66}$$

where $\mathcal{S}_j$ are the possible observed sequences. It is thus required to determine the probability for all possible sequences $\mathcal{S}_j$. As in practice it is hardly possible to determine these, Bayes' law is applied for re-formulating as follows:

$$p(\mathcal{S}_j|\underline{X}) = p(\underline{X}|\mathcal{S}_j)\frac{p(\mathcal{S}_j)}{p(\underline{X})} \tag{7.67}$$

As the probability $p(\underline{X})$ depends only on the feature vector series $\underline{X}$ and thus is independent of $\mathcal{S}_j$, it can be neglected within the maximum search over all sequences $\mathcal{S}_j$:

$$\underbrace{p(\underline{X}|\mathcal{S}_j)}_{\text{AM}} \cdot \underbrace{p(\mathcal{S}_j)}_{\text{LM}} \overset{!}{=} \max, \tag{7.68}$$

where the AM and LM represent the acoustics and semantics or syntax, and can be modelled by the sequence of audio events—in the example of continuous speech recognition these would be words, in the case of chord recognition, these would be the chords. In order to weight the influence of the LM, an exponential factor Λ—the so-called LM scaling factor—can additionally be introduced leading to:

$$p(\hat{\mathcal{S}}|\underline{X}) = \max_{\mathcal{S}_j} p(\underline{X}|\mathcal{S}_j) \cdot p(\mathcal{S}_j)^{\Lambda}. \tag{7.69}$$

The LM scaling factor is usually determined empirically or can be learnt in semi-supervised manner [37] and is often in the range of 10 ± 5.

The sequence that maximises the expression is output as best estimation $\hat{\mathcal{S}}$:

$$\hat{\mathcal{S}} = \arg\ \max_{S \in \mathcal{U}} p(\underline{X}|\mathcal{S}) \cdot p(\mathcal{S})^{\Lambda}, \tag{7.70}$$

where $\mathcal{U}$ represents all allowed sequences. Let us now assume that every sequence $\mathcal{S}_j$ is a sequence of audio events $a_1, a_2, a_3, \ldots, a_A$. In the following a single sequence $\mathcal{S}_j$ is highlighted. For this sequence then holds:

$$p(\mathcal{S}_j) = p(a_1, a_2, \ldots, a_A) \tag{7.71}$$

If we further assume that the acoustic realisations of the audio events are independent of each other, the audio events can be modelled individually:

$$p(\underline{X}|\mathcal{S}_j) = p(\underline{x}_1, \ldots, \underline{x}_i)p(\underline{x}_{i+1}, \ldots, \underline{x}_j) \ldots p(\underline{x}_{k+1}, \ldots, \underline{x}_A) \tag{7.72}$$

It is assumed that these audio events occur without pauses and pauses are treated as audio events. Note that the audio event boundaries $i, j, \ldots, k$ and audio event number A are unknown and need to be determined by the classifier.

In the same way each audio event can be constructed by a sequence of audio sub-events (ASE) one level lower in hierarchy again assuming independence. In the case of speech these could be phonemes, triphones, or syllables, etc. In the case of chord arpeggios, these could be note events. If the ASE are modelled by HMM, the Viterbi algorithm can be applied on all three layers [36]: for the search of the state sequence within the HMMs, for the sequence of the individual ASE HMMs in each audio event, and for the sequence of the audio events, i.e., $\hat{\mathcal{S}}$.

At the audio event transitions the LM can be applied to model higher level information by transition probabilities [38]. These can for example be N-grams that model the conditional probability of a sequence of consecutive audio events, e.g., two or three. The Viterbi path determines the optimal path through all layers and by that the optimal sequence recognition with the optimal sequence of audio events—for an illustrative example see 7.13 where an according 'Trellis' is shown [32].

If the number of audio events—the 'vocabulary' size—is very large, the Viterbi search can become very computationally demanding and thus slow. Though at time step t only a single column needs to be analysed in the Trellis diagram (cf. Fig. 7.13), all emission probabilities in all states for all ASE in all audio events need to be computed. In the case of large vocabulary continuous speech recognition (LVCSR) this may easily require computation of more than 100 000 normal distributions in 10 ms [36]. One can thus make use of the fact that usually many paths in the Trellis are not promising in the sense that they lead to the overall best path, which is searched for. The 'beam search' thus prunes these candidates accepting a sub-optimal solution (usually less then one percentage point increase in error probability) at considerable speed-up and reduced memory consumption. This is reached by a smart list management in five consecutive steps [39]:

First, at time step t a list of all active states is set-up. This contains all the points in the Trellis diagram whose validation exceeds a given threshold. Each element in the list is stored by the audio event number, the ASE number, the state number, and its validation.

Then, from this list all possible subsequent states are computed that can be reached by the Viterbi path-diagram. To this end, the path diagram is applied in forward direction by overwriting the place-holders in the transition from $(t-1)$ to (t) each according to higher validation. The algorithm works in a recursive manner as usual and the effect is the same as when applying the path diagrams as in the usual case in backward direction.

Next, the list of subsequent states is reduced by deleting those states below the threshold—this is the actual pruning. This threshold is best constantly adapted to the

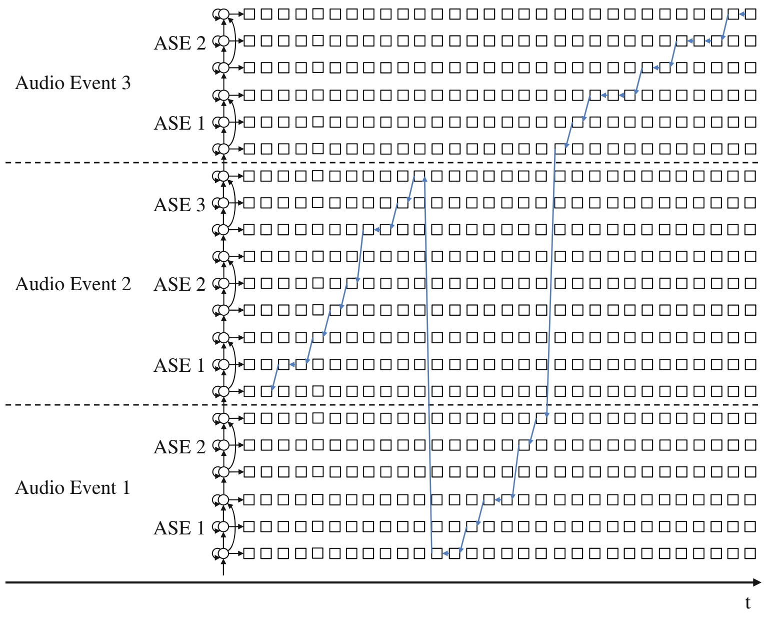


Fig. 7.13 Viterbi search of the optimal audio event sequence, Trellis diagram for the hierarchical recognition of audio events that consist of audio sub-events (ASE). The backtracking path is shown over time, and squares represent feature vector observations. HMMs (one per ASE) are shown schematically in Bakis topology. After backtracking the sequence of audio events 2, 1, 3 is recognised

current step in time. By that, the 'beam width' is broadened or narrowed according to the validation of the concurring paths' ascent or decline. This width is decisive for the trade-off between higher accuracy (broadened width) and higher speed (narrowed width).

Subsequently, at audio event transitions the value of the LM is added in the computation and it is jumped to the first state of the first model of the new audio event. In addition the required back-tracking information is stored.

Finally, the best audio event sequence is obtained at its end by the usual back-tracking, and the recognised audio events and their boundaries are output.

In practical applications, this particularly efficient search algorithm can reach reductions of the number of states to be computed of 1:1 000 [36]. The overall approach integrates knowledge of information on different levels in hierarchy to avoid early wrong decisions.

7.4 Ensemble Learning

Up to now, a number of learning algorithms was presented. In order to benefit from diverse advantages of these, one can aim at a synergistic heterogeneous combination of these. Alternatively, or in addition, homogeneous combination of the same learning algorithms, but instantiated differently, can help overcome training instability [40]. Examples of training unstable classifiers include ANNs, DTs or rule-based approaches. Overall, such combinations are known as 'ensembles' or 'committees' of learning algorithms. Owing to the increased computation effort, often so-called 'weak' classifiers are preferred in the construction of ensembles.

The aim is to reach a minimum mean square error (MSE) E of the algorithm. If the MSE is interpreted as expectation value E over all instances' feature vectors $\underline{x}$, one obtains:

$$\begin{aligned} E &= \mathrm{E}\{(\hat{y} - y)^2\} \\ &= \mathrm{Var}\{\hat{y} - y\} + \mathrm{E}\{\hat{y} - y\}^2 \\ &= \mathrm{Var}\{\hat{y}\} + \mathrm{E}\{\hat{y} - y\}^2, \end{aligned} \tag{7.73}$$

where $\hat{y}$ is the output of the learning algorithm and y is the target output.

The term $\mathrm{E}\{\hat{y} - y\}^2$ is known as square bias. It resembles the systematic deviation of the learning algorithm from the target. $\mathrm{Var}(\hat{y})$ is the variance of the output of the learning algorithm. For the minimisation of E one thus has to ideally reduce bias and variance. However, in practice, mostly only one of these two is significantly reduced in the majority of ensemble methods.

The task is thus now to construct ensembles and find a mechanism for the final decision. A simple solution for this decision is majority voting—the example in the ongoing for this type will be Bootstrap-Aggregating or Bagging for short that mainly reduces the variance. In addition, one can introduce weights for individual instances or results. This will be exemplified by Boosting, which in principle reduces both—bias and variance—however, variance to a significantly lower extent [41]. More elaborately, but requiring additional training partitions and more computational effort, one can also use a learning algorithm to train this weighting. To this end, Stacking will be introduced and an efficient example of a Tandem architecture will be shown.

7.4.1 Bootstrapping

Bagging [40] constructs ensembles of the same learning algorithm that is trained on different sub-sets of the training set $\mathcal{L}$. These sub-sets are sampled by sampling with replacement. This is the actual bootstrapping process. The cardinality of the samples is usually chosen as $|\mathcal{L}|$. Following a sampling with replacement strategy, on average 63.2 % of the training instances are covered in each sub-set, whereas the remaining percentage consists of duplicates. A variant that ensures that all samples are contained in each sub-set is called Wagging. The final decision is made by unweighted majority vote over the 'class votes' per classifier. As for regression, the mean over the results of the re-instantiated instances of the regressor is computed as final decision.

Boosting or Arcing [42] introduces a weighting for the voting (or averaging) process. Weights are chosen indirectly proportional to the error probability in order to emphasise the 'difficult cases' [43]. An option of realising weighting is to sample these instances repeatedly according to the weight. By that, the construction of ensembles follows an iterative procedure wherein the observed error probabilities are chosen by individual learning algorithms. Usually, one obtains better results as in Bagging, however, downgrades may also occur [5]. In any case, the computational effort is higher owing to the iterative procedure. One of the most popular Boosting algorithms is Adaptive Boosting, or AdaBoost for short. Adaptive refers to the iterative focus on the cases producing errors. Let $\underline{x}_l$, $l = 1, \ldots, L$ be the feature vectors in the training set $\mathcal{L}$, $L = |\mathcal{L}|$. As for SVMs or DTs, the original AdaBoost algorithm is suited only for two classes. The variant AdaBoost.M1, however, is an extension suited multiple classes M. By that, the class assignment for the instance $\underline{x}_l$ is given by $y_l \in \{1, \ldots, M\}$. To each instance $\underline{x}_l \in \mathcal{L}$ weights w_l are assigned. These are all initialised as $w_l = 1/L$ and are—as indicated—considered during computation of a weighted error measure and the training of the classifier. The core of the algorithm is now the determination of the weights β_t for the classifier with index t, where β_t depends on the error probability ε_t of the classifier.

Given the training set $\mathcal{L}$ and a number T of time steps $t = 1, \ldots, T$ the following steps are carried out:

1. A classifier with the decision $\hat{y}_t : \mathcal{X} \rightarrow \{1, \ldots, M\}$ is trained on $\mathcal{L}$ considering the weights w_l. As indicated, this can be realised by sampling a sub-set according to the weights as probability distribution.
2. The weighted classification error ε_t is computed:

$$\varepsilon_t = \sum_{l:\hat{y}_{t,l} \neq y_l} w_l. \tag{7.74}$$

3. If $\varepsilon_t > 1/2$, then repeat steps 1–3; terminate after N repetitions.
4. Else compute classifier β_t as

$$\beta_t = \begin{cases} 10^{-10} & \text{if } \varepsilon_t = 0 \\ \frac{\varepsilon_t}{1-\varepsilon_t} & \text{else} \end{cases}, \tag{7.75}$$

   where the constant 10^{-10} is arbitrarily chosen to avoid division by zero in Eq. (7.77) below.
5. If $\varepsilon_t \neq 0$, then the new weights $w_l^{'}$ as used in the following iterations result in:

$$w_l^{'} = \begin{cases} w_l \beta_t & \text{if } \hat{y}_l = y_l \\ w_l & \text{else.} \end{cases} \tag{7.76}$$

6. The weights $w_l^{'}$ are normalised for their sum to be one.

The decision $\hat{y}_{\text{Ada}}$ of the ensemble classifier is then

$$\hat{y}_{\mathrm{Ada}}(\underline{x}) = \arg\max_y \sum_{t:\hat{y}_t(\underline{x})=y} \log \frac{1}{\beta_t}. \tag{7.77}$$

Looking at Eq. (7.77), the decisions of the classifiers considered as 'strong'—i.e., those with a small β_t—is weighted higher than those of the classifiers accordingly considered as 'weak'. In particular, classifiers with a recognition rate merely above chance level will benefit most form boosting. If $\varepsilon_t \leq 1/2$ for $t = 1, \ldots, T$, one can show that for the average error $\varepsilon_{\mathrm{Ada}}$ of $\hat{y}_{\mathrm{Ada}}$ holds [43]:

$$\varepsilon_{\mathrm{Ada}} \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right), \quad \gamma_t = 1/2 - \varepsilon_t. \tag{7.78}$$

The condition $\varepsilon_t \leq 1/2$ can always be met for a two-class problem, however, for multi-class problems this is a strong limitation for weak classifiers. This can be overcome by reducing multi-class problems to multiple binary decisions such as one-versus-all, one-versus-one, half-versus-half or other groupings. An alternative is provided by the AdaBoost.M2 algorithm which integrates this formulation of multi-class problems by binary decisions—for details refer to [43].

A downside of Boosting is its susceptibility to noisy data, as mis-classified instances owing to noise may be classified correctly by chance, but are still assigned a high weight. This is for example given for problems with uncertain ground truth. Further, a high number of learning instances is usually required.

To benefit from the better minimisation of variance as in Bagging and the reduction of bias as in Boosting, these two can be combined sequentially: Usually, sub-ensembles built by AdaBoost are extended by Bagging to turn sub-ensembles into ensembles. This is often done with Wagging instead of Bagging and known as Multi-Boosting [41]—often the most efficient approach. The parameters of choice are the number and size of sub-ensembles. Usually K sub-ensembles of size K are built, resulting in K^2 instantiations of the classifier. A parallel combination is, however, not possible owing to the diverse weighting strategies of these two algorithms.

7.4.2 *Meta-Learning*

The principle of meta-learning is to unite strengths of several heterogeneous learning algorithms—now usually on the same training set. In Stacking [44], a higher level learning algorithm—the meta learner—learns literally speaking 'whom to trust when': After seeing the decisions of each lower level learning algorithm's—the base learner's— result, it comes to the final decision [45]. The meta-level is also known as level-1 and the base-level as level-0— this holds also for the type of data on these levels. On level-1, only pre-decisions are seen as input data. On level-0, the original data is seen. In order to train the level-1 learning algorithm, a J-fold cross-validation is needed (cf. Sect. 7.5.1) to assure disjoint data from training of the level-0 learning algorithms. The choice of learning algorithms for these two levels is often based on experience and exploration, as a full comprehension is still missing in the literature. However, statistical classifiers, DTs, and SVMs as introduced previously can be reasonably combined on level-0 [46]. In contrast, these seem to be less suited on level-1, where mostly Multiple Linear Regression (MLR) is chosen. MLR is different from simple linear regression only by use of multiple input variables. In the case of regression, confidences $P_{k,i}(\underline{x}) \in [0; 1]$ are assumed per base learner $k = 1, \ldots, K$, and each class $i = 1, \ldots, M$. If the level-0 classifier k only decides for exactly one class i without provision of its confidence, i.e., $\hat{y}_k = i$, the level-1 decision by MLR is as follows:

$$P_{k,i}(\underline{x}) = \begin{cases} 0 & \text{if } \hat{y}_k(\underline{x}) \neq i \\ 1 & \text{else.} \end{cases} \tag{7.79}$$

Applying non-negative weighting coefficients $\alpha_{k,i}$ per class and learner, the computation of the MLR per class i is obtained by:

$$\mathrm{MLR}_i(\underline{x}) = \sum_{k=1}^{K} \alpha_{k,i} P_{k,i}(\underline{x}). \tag{7.80}$$

During the recognition phase the class i with the highest $\mathrm{MLR}_i(\underline{x})$ is chosen for an observed unknown feature vector $\underline{x}$, i.e., the decision $\hat{y}$ is:

$$\hat{y} = \arg\max_i \mathrm{MLR}_i(\underline{x}). \tag{7.81}$$

A high value of $\alpha_{k,i}$ thus shows a high confidence in the performance of learner k for the determination of class i [40]. For the determination of the coefficients $\alpha_{k,i}$ the Lawson- and Hanson method of the least squares can be used, which will not be described here. The optimisation problem to be solved results per each learner $k = 1, \ldots, K$ in the minimisation of the following expression, in which j represents the index of the training sub-set of the J-fold cross-validation:

$$\sum_{j=1}^{J} \sum_{l=1}^{L} (y_l - \sum_{i=1}^{M} \alpha_{k,i} P_{k,i,j}(\underline{x}))^2. \tag{7.82}$$

In [45] it is shown that the meta-classification on the basis of the actual confidences of the level-0 learners results in an improvement in the majority of cases as opposed to Eq. (7.79). This is known as StackingC—short for Stacking with Confidences [46]. In [45] a description on obtaining confidence values for diverse learners is given.

Simpler alternatives use either an unweighted majority vote or one based on the mean confidences. This can also be applied in the case of regression.

Overall, ensemble learning linearly increases the computation effort. Whereas Bagging and Stacking methods can be distributed on several CPUs for parallelisation, this is not possible in the iterative Boosting process. The lowest error rate is usually obtained by StackingC, which, however, requires an extra training set for the meta-learner. It is further also suited for 'strong' learners. Finally, one can integrate Bagging and Boosting in Stacking.

7.4.3 Tandem Learning

The strengths of diverse learning algorithms can also be combined in sequential manner. An example is Tandem learning, here exemplified by a static learner that incorporates LSTM and discriminative learning abilities—namely a BLSTM RNN—, with a dynamic learner —a multi-stream HMM—that has warping abilities and 'sees' the BLSTM predictions and the original feature vectors. The structure of this multi-stream decoder can be seen in Fig. 7.14: s_t and $\underline{x}_t$ represent the HMM state and the audio feature vector, respectively, while b_t corresponds to the discrete frame-level prediction of the BLSTM network (shaded nodes). Squares denote observed nodes and white circles represent hidden nodes. In every time frame t the HMM uses two (not statistically) 'independent' observations: The audio features $\underline{x}_t$ and the BLSTM prediction feature b_t. The vector $\underline{x}_t$ also serves as input for the BLSTM, whereas the size of the BLSTM input layer i_t corresponds to the dimensionality of the audio feature vector. The vector $\underline{o}_t$ contains one probability score for each of the P different audio target classes at each time step. b_t is the index of the most likely class:

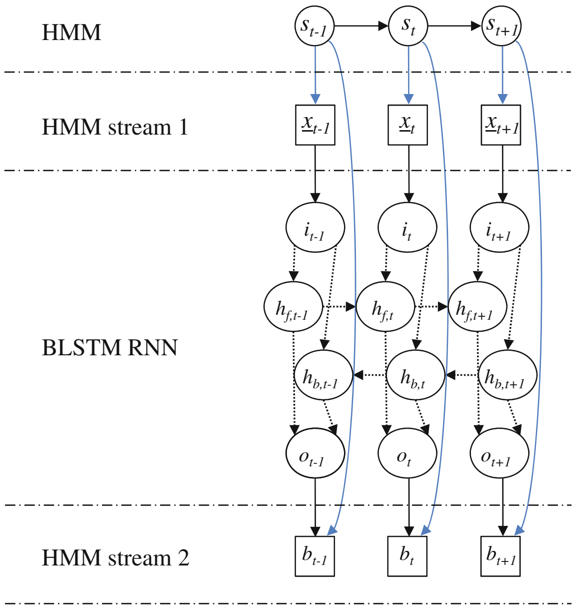


Fig. 7.14 Architecture of the multi-stream BLSTM-HMM decoder: s_t: HMM state, $\underline{x}_t$: acoustic feature vector, b_t: BLSTM class prediction feature, i_t, o_t, $h_{f,t}/h_{b,t}$: input, output, and hidden nodes of the BLSTM network; squares correspond to observed nodes, white circles correspond to hidden nodes, *shaded circles* represent the BLSTM network

$$b_t = \arg\max_j (\underline{o}_{t,1}, \ldots, \underline{o}_{t,j}, \ldots, \underline{o}_{t,P}) \tag{7.83}$$

In every time step the BLSTM generates a class prediction according to Eq. (7.83) and the HMM models $\underline{x}_{1:T}$ and $b_{1:T}$ as two independent data streams. With $\underline{y}_t = [\underline{x}_t; b_t]$ being the joint feature vector consisting of continuous audio features and discrete BLSTM observations and the variable a denoting the stream weight of the first stream (i.e., the audio feature stream), the multi-stream HMM emission probability while being in a certain state s_t can be written as

$$p(\underline{y}_t|s_t) = \left[\sum_{m=1}^{M} c_{s_t m}\mathcal{N}(\underline{x}_t; \underline{\mu}_{s_t m}, \underline{\Sigma}_{s_t m})\right]^a \times p(b_t|s_t)^{2-a}. \tag{7.84}$$

Thus, the continuous audio feature observations are modelled via a mixture of M Gaussians per state while the BLSTM prediction is modelled using a discrete probability distribution $p(b_t|s_t)$. The index m denotes the mixture component, $c_{s_t m}$ is the weight of the m'th Gaussian associated with state s_t, and $\mathcal{N}(\cdot; \underline{\mu}, \underline{\Sigma})$ represents a multivariate Gaussian distribution with mean vector $\underline{\mu}$ and covariance matrix $\underline{\Sigma}$. The distribution $p(b_t|s_t)$ is trained to model typical class confusions that occur in the BLSTM network.

7.5 Evaluation

7.5.1 *Partitioning and Balancing*

We now deal with typical ways of evaluating audio recognition systems' performance. We thereby focus on measurements that judge the reliability of the recognition result as these are of major interest in the extensive body of literature on intelligent speech, music, and sound analysis. However, as shown in the requirements section, a number of further aspects could be considered, such as real-time ability.

Evaluation should ideally be based on test partition(s) of suited audio databases that have not been 'seen' during system optimisation. Such optimisation includes data-based tuning of any steps in the chain of audio analysis including enhancement, feature extraction and normalisation, feature selection, parameter selection for the learning algorithm, etc. Thus, besides a training partition, a 'development' partition is needed for the above named optimisation steps. During the final system training, however, training and development partitions may be united in order to provide more learning material to the system. In general, one wishes all partitions to be somewhat large. For test, this is needed in order to provide significant results. Popular 'percentage splits' are thus 40 %:30 %:30 % for training, development, and test. In case of very large databases, as often given in ASR, the test partition is often chosen smaller, as around 10 %.

A solution to use as much data as possible for all partitions is the cross-validation. The overall corpus is thereby partitioned into J sets of equal size. These should be stratified, i.e., each set should show the same distribution of instances among classes or the continuum in case of numeric labels. If this is given, one speaks of J-fold stratified cross-validation (SCV). The evaluation is repeated J times with changing 'role' of the partitions. In each cycle $i = 1, \ldots, J$ partition i can for example be used as test set and the remaining ones are united for training. After J cycles, each partition has then been used for testing once, and at the same time the maximum amount of training data was provided in each cycle. The final result is then usually provided as mean of the cycles. In addition, one can provide the standard deviation or similar measures to provide an impression on the 'stability' of the alteration of the learning material and the dependence on the test partition. If one needs an additional development partition, this could, e.g., be partition $(i + 1) \bmod J$. Popular values for J are three—this allows for transparent swapping of train, develop, and test sets without too high computational effort—or ten, which is reasonable if the database is very small, and too little training data would be provided by a third of the data. In general, one usually obtains better results with increasing J, as increasingly more training material is provided. This is, however, non-linear. In the extreme case, a single instance is left out at each cycle. This is known as leave-one-out (LOO).

A number of further criteria need to be respected for partitioning of a database: For example, independence of speakers, interprets, or sound sources, i.e., in the test partition the audio should be as independent as possible depending on the task of interest. In the case of cyclic iteration, this leads to a variant of LOO, where all instances of one aspect are clustered and left out at a time. An example is Leave One Speaker Out (LOSO) in intelligent speech analysis. Next, one wishes to keep good balance of all factors throughout the partitions. In particular the development partition should be similar in its characteristics to the test one in order to optimise the system in the right way. Next, partitioning should ideally be transparent and easy to reproduce. Thus, random partitioning can be a sub-optimal choice, as one has to provide the instance list or random seed and random function in order to allow for others to reproduce the partitions. As evaluation results depend on the (optimal) partitioning, one should make the choice also straight forward, such as by partitioning by sub-sequent speaker or song ID or similar.

In many cases, instances will be highly imbalanced across classes or the number scale. This can lead to preference of the 'majority class' which is reasonable if one wants to recognise as many instances correctly as possible. However, this comes at the cost of the under-represented class, and in extreme cases, such classes are completely ignored. If it is thus of higher importance to have a good balance in the recognition, balancing of the training set instances is advisable. Note that this is not required for all learning algorithms, as many can explicitly or implicitly model the class priors in the decision process. An example is the maximum a-posteriori (MAP) strategy for statistic learners such as HMMs, where the class priors are by intention multiplied with the model's generation probability to favour the majority class. As opposed to this, the maximum likelihood estimation (MLE) principle does not use priors. For other learning algorithms shown so far, such as SVMs or DTs, this is not directly possible and balancing of instances can be the preferred option.

Three different strategies are usually employed to balance the instances in the training set [47–49]: The first is down-sampling, in which instances from the over-represented classes are randomly removed until each class contains the same number of instances. This procedure usually withdraws a lot of instances and with them valuable information, especially in highly unbalanced situations: It always outputs a training dataset size equal to the number of classes multiplied with number of instances in the class with least instances. In highly unbalanced experiments this procedure thus leads to a pathologically small training set. The second method used is up-sampling, in which instances from the classes with proportionally low numbers of instances are duplicated to reach a more balanced class distribution. This way no instance is removed from the training set and all information can contribute to the trained classifier. To not falsify the classification results, it is important that only the training instances are upsampled. Naturally, one never balances test set instances. Likewise, replacement of instances is allowed so that equal class distribution is also achievable in highly unbalanced experiments. At the same time, it is ensured that each original instance is preserved in the training material. A mixed up-, and down-sampling strategy can be also be followed where instances from the majority class are deleted and from the minority class(es) are multiplied. This compromise keeps the overall number of instances at reasonable size, as with sheer up-sampling the problem of learning may become computationally too expensive. A third variant is assignment of different weighting of instances for the computation of the classifier objective function. In practice, this is often actually often solved by classifier internal up-sampling, and may lead to less stable results, while not providing any advantage in our respect, as obtainable performances are not higher, which is why this variant is not further pursued in this book. However, this may be well of interest in an on-line system which needs to be adapted, e.g., when a user labels a new song to adapt his audio-playing device. The latter are known as 'cost-sensitive' approaches where one 'punishes' confusions that should not occur in the case of discrete classes.

The question remains, how to pick the instances that are multiplied in the training set or deleted from it. While one can inject random into the selection process, this contradicts the above requested transparency and reproducibility of experiments by others. An easy strategy that does, however, not provide perfect balance, is thus the use of integer up-sampling factors for the minority classes. In addition, there are specialised algorithms that attempt to balance instances in an intelligent way. The idea is to up- or downsample those instances which are of particular relevance, as they are the 'hard' and 'interesting' cases and should not be emphasised on or at least not lost. An example of such an approach is the Synthetic Minority Over-sampling Technique [50].

7.5.2 Evaluation Measures

In the following, evaluation criteria for classifiers are considered at first. These will be followed by such for regressors where a continuous relation between the output of the learning algorithm and the target needs to be evaluated. In the case of classification, however, we need to compare discrete predicted class labels and compare these with the ground truth target. For simplification— without limitation of the general case— let the rejection class be assumed to be inherently modelled, i.e., rejection is one of the target classes. By that, we can consider the classification task as a mapping $\mathcal{X} \to \{1, \ldots, C\}, \underline{x} \mapsto \hat{y}$.

Evaluation criteria are defined as related to the test set's $\mathcal{T}$ instances, and the individual instances are each assigned to exactly one target class $i \in \{1, \ldots, C\}$:

$$\mathcal{T} = \bigcup_{i=1}^{C} \mathcal{T}_i = \bigcup_{i=1}^{C} \{\underline{x}_{i,n} \mid n = 1, \ldots, T_i\}, \tag{7.85}$$

where T_i is the number of instances in the test set that belong to class i. By that, the test set has the size $|\mathcal{T}| = \sum_{i=1}^{C} T_i$. Note, however, that attempts exist to find evaluation criteria where several classes may be assigned to one instance. This requirement is for example given in the case of the classification of a speaker's emotion, where one is not only 'surprised', but e.g., 'happily surprised' or 'angrily surprised' which led to the introduction of soft emotion profiles [51]. Similarly, music genre or ballroom dance style are often ambiguous in music analysis, cf. musical pieces that allow for either Rhumba or Foxtrott as choice of dance.

We will first consider evaluation measures for classification in the general case of two or more classes (i.e., $M \geq 2$) [1]. The most common measure is the probability that an instance of the test set is classified correctly. This is usually referred to as (weighted) accuracy *WA*, or weighted average recall or recognition rate.

$$\begin{aligned} \mathrm{WA} &= \frac{\#\,\text{correctly classified test instances}}{\#\,\text{test instances}} \\ &= \frac{\sum_{i=1}^{M} \left|\left\{\underline{x} \in \mathcal{T}_i \mid \hat{y} = i\right\}\right|}{|\mathcal{T}|}. \end{aligned} \tag{7.86}$$

If this rate is given per class i, one speaks of the class-specific recall RE_i:

$$\mathrm{RE}_i = \frac{\left|\left\{\underline{x} \in \mathcal{T}_i \mid \hat{y} = i\right\}\right|}{T_i}. \tag{7.87}$$

With $p_i = T_i / |\mathcal{T}|$ as the prior probability of class i in the test set further holds:

$$\mathrm{WA} = \sum_{i=1}^{M} p_i \, \mathrm{RE}_i. \tag{7.88}$$

The weighting by p_i in Eq. (7.88) leads to the name weighted accuracy. A special case of the calculation of accuracy is the word accuracy as encountered in the recognition of continuous speech. This accuracy is calculated by consideration of three types of errors: deletion, insertion, and substitution of words. With D, I, and S being the numbers for each type of these errors and N being the number of words in the test set, the word accuracy $\mathrm{WA}_{\mathrm{words}}$ is obtained by:

$$\mathrm{WA}_{\mathrm{words}} = \frac{N - D - I - S}{N}. \tag{7.89}$$

Note that, $N - (S + D)$ would be the number of correctly recognised words. Further, dynamic alignment of the recogniser output string and the reference transcription is needed to decide on the minimal number of errors, as a substitution could be counted as a deletion plus an insertion. This is also known as shortest Levenshtein distance. As word accuracy is also a type of WA—the accuracy depends on the frequency of occurrence of a specific word in the test set—it is also referred to as WA in the ongoing. However, from the context it will be clear that it is computed as word accuracy.

If balance of instances among classes is (highly) unbalanced, one can prefer to exchange the priors p_i for all classes by the constant weight $\frac{1}{M}$. This is known as unweighted accuracy *UA* or unweighted average recall:

$$\mathrm{UA} = \frac{\sum_{i=1}^{M} \mathrm{RE}_i}{M}. \tag{7.90}$$

The numerator in Eq. (7.87) equals the number of instances in $\mathcal{T}$, for which the decision was correctly made for class i. This is the number of 'true positives' TP_i as opposed to the false positives *FP* for class i:

$$\mathrm{FP}_i = \left|\left\{\underline{x} \in \mathcal{T} - \mathcal{T}_i \mid \hat{y} = i\right\}\right|. \tag{7.91}$$

With *TP* and *FP* we can define the precision *PR*:

$$\mathrm{PR}_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FP}_i}. \tag{7.92}$$

As increasing RE_i may come at the cost of decreasing PR_i, as many instances are assigned by mistake to class i, the wish for a measure that unites these two arises. This is given by their harmonic mean, known as F_1-measure (the subscript '1' is used for equal weighting of recall and precision—other common weights are doubling one up, i.e., F_2- or $F_{\frac{1}{2}}$-measure:

$$F_{1,i} = 2\frac{\mathrm{RE}_i \mathrm{PR}_i}{\mathrm{RE}_i + \mathrm{PR}_i}. \tag{7.93}$$

Considering decisions against class i, one can further introduce 'true negatives' TN_i and 'false negatives' FN_i:

$$\mathrm{TN}_i = \left|\left\{\underline{x} \in \mathcal{T} - \mathcal{T}_i \mid \hat{y} \neq i\right\}\right|, \tag{7.94}$$

$$\mathrm{FN}_i = \left|\left\{\underline{x} \in \mathcal{T}_i \mid \hat{y} \neq i\right\}\right|. \tag{7.95}$$

It is further of interest to investigate which classes are 'confused' with which. The according 'confusion matrix' $\underline{C} = (c_{i,j})$ thus has the entries:

$$c_{i,j} = \left|\left\{\underline{x} \in \mathcal{T}_i \mid \hat{y} = j\right\}\right|. \tag{7.96}$$

This matrix $\underline{C}$ contains all named measures as follows:

$$\mathrm{WA} = \frac{\mathrm{tr}(\underline{C})}{|\mathcal{T}|}, \tag{7.97}$$

$$\mathrm{RE}_i = \frac{c_{i,i}}{T_i} = \frac{c_{i,i}}{\sum_{j=1}^{M} c_{i,j}}, \tag{7.98}$$

$$\mathrm{PR}_i = \frac{c_{i,i}}{\sum_{j=1}^{M} c_{j,i}}, \tag{7.99}$$

$$\mathrm{TP}_i = c_{i,i}, \tag{7.100}$$

$$\mathrm{FP}_i = \sum_{i \neq j} c_{j,i}. \tag{7.101}$$

In the case of binary decisions, the term TP_1/T_1 corresponds to the detection probability or 'true positive rate' (TPR), and FP_1/T_2 to false alarm probability or 'false positive rate' (FPR). Graphical evaluation often makes use of the Receiver Operating Characteristic (ROC, TPR vs. FPR) or its alternative, the Detection Error Trade-off (DET, false negative rate vs. FPR) curve. Such a plot demands for multiple evaluations of the learning algorithm's model or knowledge of confidences per instance in order to adjust a threshold for curve plotting. Popular measures to represent the plots in a single number are the 'area under the curve' (AUC) or the 'equal error rate' (EER). In case of more than two classes, i.e., $M > 2$, these measures are usually given as per one-versus-all.

We now shift to evaluation criteria for continuous value estimation, i.e., regression. Again, these are defined as related to the test set's $\mathcal{T}$ instances, and the individual instances are now each assigned to a continuous value $\hat{y} \in \mathbb{R}$. The test set has the size $|\mathcal{T}|$. In the case of regression, the common evaluation measure is the Pearson's correlation coefficient CC:

$$\mathrm{CC} = \frac{\sum_{n=1}^{|\mathcal{T}|}\left(\hat{y}_n - \overline{\hat{y}}\right)(y_n - \overline{y})}{\sqrt{\sum_{n=1}^{|\mathcal{T}|}\left(\hat{y}_n - \overline{\hat{y}}\right)^2 \cdot \sum_{n=1}^{|\mathcal{T}|}(y_n - \overline{y})^2}}, \tag{7.102}$$

with the averages

$$\bar{\hat{y}} = \frac{1}{|\mathcal{T}|} \sum_{n=1}^{|\mathcal{T}|} \hat{y}_n, \tag{7.103}$$

and

$$\bar{y} = \frac{1}{|\mathcal{T}|} \sum_{n=1}^{|\mathcal{T}|} y_n. \tag{7.104}$$

In addition, the the Mean Linear Error (MLE)—often referred to as Mean Absolute Error *MAE*—can be given:

$$\mathrm{MAE} = \frac{1}{|\mathcal{T}|} \sum_{n=1}^{|\mathcal{T}|} |\hat{y}_n - y_n|, \tag{7.105}$$

MAE can be very intuitive, such as in the case of age determination in years of a speaker. Then, the MAE would be the absolute error in years, by which the regressor is mistaken 'on average'. However, in case of tasks where relative difference is more important than absolute numbers and the gold standard is less certain, such as for likability of a speaker or interest of a speaker on a continuous scale, CC is usually more representative and has a minimum and maximum independent of the task. CC is thus written without a leading zero in this book for better readability. This is different for MLE and MAE, as the number range varies.

As a general remark, it is important to note that all these evaluation measures naturally depend on the choice of the test instances. Apart from that, meaningful significance analyses should be considered as the difference between two results also depends on the quantity of test instances [52, 53]. Frequently employed tests contain, e.g., the one-sided z-test [54], which is the preferred choice in this book, and the common level of 0.05 is the minimum requirement for the claim of significance. Note that usually significance tests base on the independence assumption of tests [55]. As a consequence, this would require different data-sets for testing. However, as the test set is typically kept fixed in this field of research, the premise to reject the null hypothesis is comparably strict [55].

References

1. Kroschel, K., Rigoll, G., Schuller, B.: Statistische informationstechnik, 5th edn. Springer, Berlin (2011)
2. Quinlan, JR., C4.5: Programs for machine learning. Morgan Kaufmann, Burlington (1993)
3. Quinlan, J.: Learning efficient classification procedures and their application to chess end games. In machine learning: an artificial intelligence approach, pp .106–121. Tioga Publishing, Palo Alto (1983)
4. Quinlan, J.: Simplifying decision trees. Int. J. Man Mach. Stud. **27**, 221–234 (1987)

5. Quinlan, J., Bagging, Boosting and C4.5. In Proceedings 14th National Conference on AI, vol. 5, pp. 725–730, AAAI Press, Menlo Park (1996)
6. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
7. Hochreiter, S., Mozer, M., Obermayer, K.: Coulomb classifiers: generalizing support vector machines via an analogy to electrostatic systems. Adv. Neural Inf. Process. Sys. **15**, (2002)
8. Cristianini, N., Shawe-Taylor, J .: An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge (2000)
9. Platt, J.: Sequential minimal optimization: a fast algorithm for training support vector machines. Technical report MSR-98–14, Microsoft Research, New York (1998)
10. Schölkopf, B., Smola, A.: Learning with kernels: support vector machines, regularization, optimization, and beyond (Adaptive computation and machine learning). MIT Press, Cambridge (2002)
11. Yang, H., Xu, Z., Ye, J., King, I., Lyu, M.: Efficient sparse generalized multiple kernel learning. IEEE Trans. Neural Netw. **22**(3), 433–446 (2011)
12. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. J. Mach. Learn. Res. **2**, 419–444 (2002)
13. Smola, A., Schölkopf, B.: A tutorial on support vector regression. Stat. Comput. **14**(3), 199–222 (2004)
14. Niemann, H.: Klassifikation von mustern. published online, 2nd, revised and extended edition (2003)
15. McCulloch, W., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin Math. Biophy. **5**, 115–133 (1943)
16. Schuller, B.: Automatische emotionserkennung aus sprachlicher und manueller interaktion. Doctoral thesis, Technische Universität München, Munich (2006)
17. Rigoll, G.: Neuronale netze. Expert-Verlag (1994)
18. Deller, J., Proakis, J., J. Hansen.: Discrete-time processing of speech signals. Macmillan Publishing Company, New York (1993)
19. Rumelhart, D., Hinton, G., Williams, R.: Learning internal representations by error propagation. In parallel distributed processing: explorations in the microstructure of cognition, vol. 1, pp. 318–362. MIT Press, Boston (1987)
20. Schalkoff, R.:Artificial neural networks. McGraw-Hill, New York (1994)
21. Riedmiller, M., Braun, H.: Rprop—A fast adaptive learning algorithm. In Proceedings of the International Symposium on Computer and Information Science, vol. 7, (1992)
22. Lacoste, A., Eck, D.: Onset detection with artificial neural networks. In MIREX (2005)
23. Werbos, P.: Backpropagation through time: what it does and how to do it. Proc. IEEE **78**(10), 1550–1560 (1990)
24. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: Kremer, S.C., Kolen, J.F. (eds.) A field guide to dynamical recurrent neural networks, pp. 1–15. IEEE Press, New York (2001)
25. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
26. Gers, F., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(10), 2451–2471 (2000)
27. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. **18**(5–6), 602–610 (2005)
28. Graves, A.: Supervised sequence labelling with recurrent neural networks. Ph.D thesis, Technische Universität München, Munich (2008)
29. Wöllmer, M., Schuller, B., Batliner, A., Steidl, S., Seppi, D.: Tandem decoding of children's speech for keyword detection in a child-robot interaction scenario. ACM Transactions on Speech and Language Processing, (Special Issue Speech Lang. Process. Children's Speech Child-mach. Interact. Appl.). vol. 7(4), August 2011, 22 pages
30. Wöllmer, M., Schuller, B., Eyben, F., Rigoll, G.: Combining long short-term memory and dynamic bayesian networks for incremental emotion-sensitive artificial listening. IEEE J. Sel. Top. Sig. Process. (Special Issue Speech Process. Nat. Interact. Intell. Environ). **4**(5), 867–881 (October 2010)

31. Wöllmer, M., Blaschke, C., Schindl, T., Schuller, B., Färber, B., Mayer, S., Trefflich, B.: Online driver distraction detection using long short-term memory. IEEE Trans. Intell. Transp. Syst. **12**(2), 574–582 (2011)
32. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proc. IEEE **77**, 257–286 (1989)
33. O'Shaughnessy, D.: Speech communication. Adison-Wesley, 2nd edn, Boston (1990)
34. Jelinek, F.: Statistical methods for speech recognition. MIT Press, Cambridge (1997)
35. Baum, L.E., Petrie, T., Soules, G., Weiss, N.: A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. Ann. Math. Stat. **41**(1), 164–171 (1970)
36. Ruske, G.: Automatische spracherkennung, 2nd edn. Methoden der Klassifikation und Merkmalsextraktion, Oldenbourg (1993)
37. White, C.M., Rastrow, A., Khudanpur, S., Jelinek, F.: Unsupervised Estimation of the Language Model Scaling Factor. In Proceedings of the Interspeech, pp. 1195–1198, Brighton (2009)
38. Furui, S.: Digital speech processing: synthesis, and recognition, 2nd edn. Signal Processing and Communications. Marcel Denker Inc., New York (1996)
39. Lowerre, B.: The harpy speech recognition system. Ph.D thesis, Carnegie Mellon University, Pittsburgh (1976)
40. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
41. Webb, G.: Multiboosting: A technique for combining boosting and wagging. Mach. Learn. **40**, 159–198 (2000)
42. Valinat, L.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)
43. Freund, Y., Schapire, R.: Experiments with a New Boosting Algorithm, pp. 148–156. In Proceedings of the International Conference on, Machine Learning (1996)
44. Wolpert, D.: Stacked generalization. Neural Netw. **5**, 241–259 (1992)
45. Ting, K., Witten, I.: Issues in Stacked Generalization. J. Artif. Intell. Res. **10**(1), 271–289 (Jan. 1999)
46. Seewald, A.: Towards understanding stacking—Studies of a general ensemble learning scheme. Ph.D thesis, Technische Universität Wien, Vienna (2003)
47. Schuller, B.: Steidl, S., Batliner, A.: The interspeech 2009 emotion challenge. In Proceedings INTERSPEECH 2009, 10th Annual Conference of the International Speech Communication Association, pp. 312–315, Brighton, September ISCA, ISCA (2009)
48. Schuller, B., Müller, R., Eyben, F., Gast, J., Hörnler, B., Wöllmer, M., Rigoll, G., Höthker, A., Konosu, H.: Being bored? recognising natural interest by extensive audiovisual integration for real-life application. Image Vision Comput. (Special Issue Visual Multimodal Anal. Human Spontaneous Behav). **27**(12), 1760–1774 (2009)
49. Schuller, B., Schenk, J., Rigoll, G., Knaup, T.: "the godfather" versuss. "chaos": Comparing linguistic analysis based on online knowledge sources and bags-of-n-grams for movie review valence estimation. In Proceedings 10th International Conference on Document Analysis and Recognition, ICDAR 2009, pp.858–862, IAPR, IEEE, Barcelona (2009)
50. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
51. Mower, E., Mataric, M., Narayanan, S.: A framework for automatic human emotion classification using emotional profiles. IEEE Trans. Audio Speech Lang. Process. **19**(5), 1057–1070 (2011)
52. Rozeboom, W.: The fallacy of the null-aypothesis significance test. Psychol. Bull. **57**, 416–428 (1960)
53. Nickerson, R.S.: Null hypothesis significance testing: a review of an old and continuing controversy. Psychol. Bull. **5**, 241–301 (2000)
54. Eysenck, H.: The concept of statistcal significance and the controversy about one-tailed tests. Psychol. Bull. **67**, 269–271 (1960)
55. Gillick, L., Cox, S.J.: Some statistical issues in the comparison of speech recognition algorithms. In Proceedings International Conference on Audio Speech and Signal Processing (ICASSP), vol. 1, pp. 23–26, IEEE, Glasgow (1989)