Chapter 5
Statistical Methods for Scene and Event Classification

Brian McFee

Abstract This chapter surveys methods for pattern classification in audio data. Broadly speaking, these methods take as input some representation of audio, typically the raw waveform or a time-frequency spectrogram, and produce semantically meaningful classification of its contents. We begin with a brief overview of statistical modeling, supervised machine learning, and model validation. This is followed by a survey of discriminative models for binary and multi-class classification problems. Next, we provide an overview of generative probabilistic models, including both maximum likelihood and Bayesian parameter estimation. We focus specifically on Gaussian mixture models and hidden Markov models, and their application to audio and time-series data. We then describe modern deep learning architectures, including convolutional networks, different variants of recurrent neural networks, and hybrid models. Finally, we survey model-agnostic techniques for improving the stability of classifiers.

Keywords Machine learning • Statistical modeling • Classification • Discriminative models • Generative models • Deep learning • Convolutional neural networks • Recurrent neural networks • Hidden Markov models • Bayesian inference

5.1 Introduction

This chapter provides an overview of machine learning methods for pattern classification. Throughout this chapter, our objective is to design algorithms which take as input some representation of an audio signal, and produce some semantically meaningful output, e.g., a categorical label indicating the presence of an acoustic event in the audio signal.

The treatment of topics in this chapter will be relatively superficial: our goal is to provide a high-level overview of methods for pattern classification, not an in-depth

B. McFee (✉)
Center for Data Science, New York University, 60 5th Ave., New York, NY 10003, USA
e-mail: brian.mcfee@nyu.edu

T. Virtanen et al. (eds.), *Computational Analysis of Sound Scenes and Events*,
DOI 10.1007/978-3-319-63450-0_5

survey of advanced statistics and machine learning. We will assume familiarity with linear algebra, multivariate calculus, and elementary probability theory and statistics. We will not cover computational learning theory or optimization, but references for those concepts will be provided.

The remainder of this chapter is structured as follows. Section 5.1 describes the fundamentals and practical considerations of statistical learning. Section 5.2 introduces discriminative models for binary and multi-class prediction problems, with a focus on linear models. Section 5.3 covers generative models, unsupervised learning, and Bayesian inference, focusing on Gaussian mixture models and hidden Markov models for audio applications. Section 5.4 provides an overview of deep learning, including multi-layer perceptrons, one- and two-dimensional convolutional networks, various formulations of recurrent neural networks, and hybrid architectures. Section 5.5 describes some useful techniques to improve the robustness and stability of classifiers. Finally, Sect. 5.6 concludes with pointers to further readings on advanced topics.

Throughout this chapter, the input representation of audio is generally left abstract, and may correspond to a summary of an entire recording or more localized representations of individual audio frames. The fundamentals of binary and multi-class discriminative classifiers described in Sect. 5.2 apply to both of these cases. For example, a static acoustic scene classification system could apply a multi-class discriminative classifier to a feature vector representing the acoustic properties of the entire audio recording, resulting in a single categorical label predicted for the entire recording. Similarly, a clip-level tagging system could apply several binary classifiers to predict the presence of multiple concepts within a recording (e.g., *speech*, *bird song*, *footsteps*), but without localizing them in time. By contrast, *dynamic* prediction tasks, such as sound event detection, would operate on localized representations (e.g., individual frames) to produce a time-series of predictions. Methods for exploiting temporal structure are described in Sect. 5.3.5 (Hidden Markov models) and Sects. 5.4.3 and 5.4.4 (convolutional and recurrent networks).

5.1.1 Preliminaries

Input data will be generically denoted as $x \in \mathcal{X}$, and output variables will be denoted as $y \in \mathcal{Y}$. The input domain $\mathcal{X}$ and output space $\mathcal{Y}$ will be left abstract for much of this chapter, but it may be helpful to think of the concrete case where $\mathcal{X} = \mathbb{R}^d$ corresponds to some pre-computed frame-level features (e.g., mel-scaled power spectra as described in Chap. 4) and $\mathcal{Y} = \{-1, +1\}$ are binary categorical labels. Input–output pairs are assumed to be jointly distributed according to some (unknown) probability distribution $(x, y) \sim \mathcal{D}$; for brevity, we will sometimes write $z = (x, y)$ to denote a labeled example. A classifier (or, more generally, a predictor) will map an observed input x to an output (label) y, and be denoted as $h : \mathcal{X} \to \mathcal{Y}$. Finally, we will characterize the accuracy of a predictor by using *loss functions*, denoted by $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, to compare an estimated label $h(x)$ to a true label y. Small values of ℓ indicate high accuracy, and high values of ℓ indicate low accuracy.

This chapter is primarily concerned with the *supervised learning* model, wherein a sample of labeled points $S = \{(x_i, y_i)\}_{i=1}^{n}$ (the *training set*) are independently and identically distributed (*I.I.D.*) by a probability distribution $\mathscr{D}$, and used to estimate the parameters of the algorithm. In general, we would like to find a predictor h that minimizes the *risk*[1]:

$$\mathbf{E}_{\mathscr{D}}\left[\ell(h(x),\, y)\right] = \int_{x,y} \ell(h(x),\, y) \times \mathbf{P}_{\mathscr{D}}[x, y]\mathrm{d}x\mathrm{d}y. \tag{5.1}$$

Put plainly, (5.1) captures the expected error rate of a predictor h over the data distribution $\mathscr{D}$. When ℓ is the 0–1 loss:

$$\ell(y, y') := \begin{cases} 0 & y = y' \\ 1 & y \neq y' \end{cases} \tag{5.2}$$

then (5.1) is the probability of incorrectly classifying a randomly selected input x. Since $\mathscr{D}$ is generally unknown, minimizing (5.1) over choices of h is not possible. The supervised learning approach is to approximate (5.1) by the *empirical risk* estimated over the sample:

$$\frac{1}{n}\sum_{i=1}^{n} \ell\left(h(x_i),\, y_i\right) \approx \mathbf{E}_{\mathscr{D}}\left[\ell(h(x),\, y)\right]. \tag{5.3}$$

The learning problem therefore amounts to minimizing an objective function (5.3) to solve for h over some class of models.

The predictor h is generally defined in terms of parameters $\theta \in \Theta$, which we denote as $h(x \mid \theta)$. Thus, the learning problem can be generally stated as minimizing (5.3) over the choice of θ from a space Θ of possible configurations:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i \mid \theta), y_i). \tag{5.4}$$

When ℓ is continuous and differentiable—such as in least-squares regression, where $\ell(y, y') = \|y - y'\|^2$—then (5.4) can be solved by iterative methods such as gradient descent, or occasionally in closed form. However, for classification problems, ℓ is often discontinuous or non-differentiable; for example, the 0–1 loss (5.2) is neither continuous nor differentiable with respect to θ. In these cases, exactly optimizing (5.4) can be a difficult computational problem [45, 74]. As a result, it is common to replace the exact loss function ℓ with a *surrogate function* f that is amenable to efficient optimization: typically this means that f is continuous and (at least piece-wise) differentiable.

[1]The notation $\mathbf{P}_{\mathscr{D}}$ denotes the probability mass (or density) with respect to distribution $\mathscr{D}$, and $\mathbf{E}_{\mathscr{D}}$ denotes the expectation with respect to distribution $\mathscr{D}$.

Surrogate objective functions may operate not directly upon the predicted label $h(x \mid \theta)$, but on some convenient, related quantity such as conditional probability of a category given the observed x. In general, we will denote the surrogate loss as a function $f : \mathscr{X} \times \mathscr{Y} \times \Theta \to \mathbb{R}_+$. To summarize, this chain of steps leads to a general formulation of learning:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} f(x_i, y_i \mid \theta), \tag{5.5}$$

where minimizing (5.5) approximately minimizes (5.4), which in turn approximates the risk (5.3) which we would ideally minimize.[2]

Finally, one may wish to encode some preferences for certain configurations of θ over others. This can be achieved by including a *regularization function* or *penalty term* $g : \Theta \to \mathbb{R}_+$ which takes low values for preferred configurations and high values for undesirable configurations. The regularized learning objective then takes the general form we will use for the remainder of this chapter:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} f(x_i, y_i \mid \theta) + g(\theta). \tag{5.6}$$

As we will see in Sect. 5.3, the general form of (5.6) can also be used for maximum a posteriori Bayesian inference, where the g term takes the role of the prior distribution on the model parameters.

5.1.2 Validation and Testing

The previous section outlines a generic recipe for building predictive models:

1. Collect a labeled training sample S,
2. Specify a surrogate loss function f and penalty g,
3. Solve (5.6) to find parameters θ^*,
4. Deploy the resulting model $h(\cdot \mid \theta^*)$.

In practice, before deploying the model θ^*, we would also like to have an estimate of how well it performs on unseen data drawn from $\mathscr{D}$. This can be estimated by using a second independent sample $S_T \sim \mathscr{D}$ known as the *test set*, which is only used for evaluating θ^* and not for parameter estimation.

[2]Quantifying the relationships between (5.5), (5.4), and (5.3) lies within the purview of statistics and computational learning theory, and is beyond the scope of this text. We refer interested readers to [48, 106] for an introduction to the subject.

By the same token, it is common for practitioners to develop multiple candidate models, generally derived from different choices of (f, g). Before moving on to testing and deployment, the practitioner must choose a particular model from among the candidates. This process is commonly referred to as *validation* or *hyper-parameter optimization*. It is important to note that the test set S_T cannot be used for validation. Any sample data which influences the selection of θ must be interpreted as *training data*, regardless of whether it appears in (5.6).

The typical approach to validation is to randomly partition the training set S into two disjoint sets S', S_V. The subset S' is used to optimize the parameters θ_{fg} for a given model specification (f, g). The complementary subset S_V, sometimes called the *validation set*, is used to estimate the risk of θ_{fg}:

$$\mathbf{E}_{\mathscr{D}}[\ell(h(x \mid \theta_{fg}), y)] \approx \frac{1}{|S_V|} \sum_{(x_i, y_i) \in S_V} \ell(h(x_i \mid \theta_{fg}), y_i). \tag{5.7}$$

This partitioning process is typically repeated several times and the results are averaged to reduce the variance of (5.7) introduced by sub-sampling the data. The validation procedure then selects θ_{fg} which achieves the lowest (average) validation error.

There are virtually countless variations on this validation procedure, such as cross-validation, stratified sampling, parameter grid search, and Bayesian hyper-parameter optimization [12, 13, 110]. A full survey of these techniques is beyond the scope of this chapter, but for our purposes, it is important to be comfortable with the concepts of validation, hyper-parameter optimization, and testing.

5.2 Discriminative Models

This section provides an overview of discriminative approaches to classification. Models will be described in terms of their objective functions, but we will omit the details of implementing specific optimization algorithms for parameter estimation.

In simple terms, a *discriminative model* seeks to predict a label y as a function of an input observation x, but does not explicitly model the input space $\mathscr{X}$. In this sense, discriminative models are simpler than *generative models* (Sect. 5.3), which must model the joint distribution over $\mathscr{X} \times \mathscr{Y}$. We will begin with an overview of binary linear models, extend them to multi-class models, and discuss their application to time-series data.

5.2.1 Binary Linear Models

The simplest models that practitioners regularly encounter are *linear models*. For binary classification problems with $\mathscr{X} \subseteq \mathbb{R}^d$, a linear model is parameterized by a weight vector $w \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$, so that $\theta = (w, b)$. The model is linear in the sense that the parameters w and b interact with the data through an inner product (and scalar addition) to produce a score $\langle w, x\rangle + b$. The output space is defined as $\mathscr{Y} = \{-1, +1\}$, so that the decision rule takes the form:

$$h(x \mid \theta) = \operatorname{sign}(\langle w, x\rangle + b), \tag{5.8}$$

and the typical loss function of interest is the 0–1 loss. As mentioned in Sect. 5.1.1, the 0–1 loss is difficult to optimize directly, and different choices of surrogate functions lead to different models and algorithms.

5.2.1.1 Support Vector Machines

One of the simplest surrogate functions for the 0–1 loss is the *margin hinge loss*:

$$f_+(x, y \mid \theta) := \max\left(0, 1 - y\left(\langle w, x\rangle + b\right)\right), \tag{5.9}$$

which incurs 0 loss when the score $\langle w, x\rangle + b$ has the same sign as y—so that the prediction is correct—and its magnitude is at least 1 (the *margin*). The choice of 1 for the margin coincides with the error for misclassification $\ell(0, 1) = \ell(1, 0)$, and ensures that f_+ provides an upper bound on the 0–1 loss as illustrated in Fig. 5.1.

Combined with a quadratic penalty on w, the hinge loss gives rise to the standard linear *support vector machine* (SVM) [30]:

$$\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\langle w, x_i\rangle + b\right)\right). \tag{5.10}$$

The hyper-parameter $\lambda > 0$ balances the trade-off between accuracy (minimizing loss) and model complexity (minimizing the norm of w).

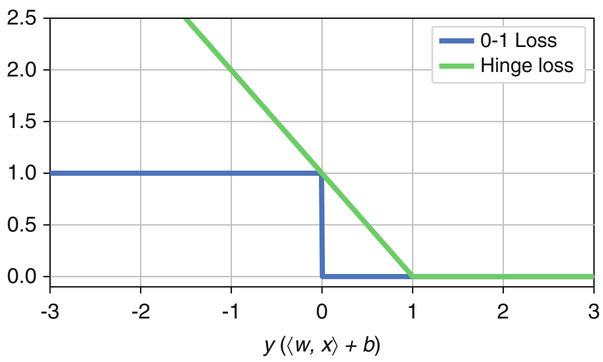

Fig. 5.1 The 0–1 loss and the hinge loss with a margin of 1. The hinge loss provides a continuous, convex upper bound on the 0–1 loss

5.2.1.2 Logistic Regression

An alternative to the SVM method in the previous section is to suppose a probabilistic model of the conditional probability $\mathbf{P}_\theta[y = +1 \mid x]$. Since the output space is binary, the Bernoulli distribution is a natural choice here:

$$\begin{aligned} \mathbf{P}[y = +1] &:= p \\ \mathbf{P}[y = -1] &:= 1 - p = 1 - \mathbf{P}[y = +1] \end{aligned} \tag{5.11}$$

where $p \in [0, 1]$ is the probability of a positive label. To parameterize a Bernoulli distribution $\mathbf{P}_\theta[y = +1 \mid x]$ by the linear score function $\langle w, x\rangle + b$, the score can be mapped to the unit interval [0, 1] via the *logistic function*:

$$\sigma(t) := \frac{1}{1 + \mathrm{e}^{-t}}. \tag{5.12}$$

This results in the following conditional distribution for the label y given the input x:

$$\begin{aligned} \mathbf{P}_\theta[y = +1 \mid x] &:= \sigma(\langle w, x\rangle + b) = \frac{1}{1 + \mathrm{e}^{-\langle w,x\rangle - b}} \\ \mathbf{P}_\theta[y = -1 \mid x] &:= 1 - \mathbf{P}_\theta[y = +1 \mid x]. \end{aligned} \tag{5.13}$$

As depicted in Fig. 5.2, the decision rule (5.8) coincides with choosing the most probable label under this model.

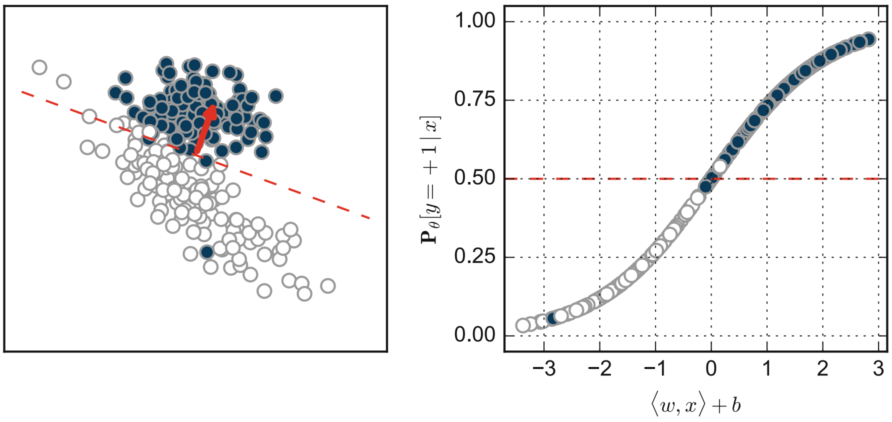

Fig. 5.2 An example of logistic regression in two dimensions. The *left plot* illustrates the data (*white* and *blue points*) and the learned linear model w (*red arrow*). The *right plot* illustrates the linear score $\langle w, x\rangle + b$ for each point x compared to the model probability $\mathbf{P}_\theta[y = +1 \mid x]$, where each point is *colored* according to its label. The decision threshold (0.5) is drawn in *red*

Taking the negative logarithm of (5.13) results in the following surrogate function:

$$f_\sigma(x_i, y_i \mid \theta) := \begin{cases} \log\left(1 + \mathrm{e}^{-\langle w, x_i\rangle - b}\right) & y_i = +1 \\ \langle w, x_i\rangle + b + \log\left(1 + \mathrm{e}^{-\langle w, x_i\rangle - b}\right) & y_i = -1 \end{cases}$$

$$= \left(\frac{1 - y_i}{2}\right)(\langle w, x_i\rangle + b) + \log\left(1 + \mathrm{e}^{-\langle w, x_i\rangle - b}\right). \tag{5.14}$$

Because of its use of the logistic function in defining (5.13), this formulation is known as *logistic regression* [32].

Although logistic regression and SVM share the same parameterization and have equivalent prediction rules, there are some key distinctions to keep in mind when deciding between the two methods. First, the scores produced by logistic regression have a natural probabilistic interpretation, whereas the SVM's scores do not directly correspond to a probabilistic model.[3] Probabilistic interpretation can be useful when the classifier must produce confidence-rated predictions, or be integrated with larger models, such as the hidden Markov models discussed later in Sect. 5.3.5. Second, the choice of regularization $g(w)$ can have significant influence on the behavior of the model. While SVMs use the ℓ_2 (quadratic) penalty, logistic regression is often implemented with ℓ_1 or ℓ_2 penalties. The ℓ_2 penalty can be seen as limiting the influence of any single feature, while ℓ_1 can be seen as encouraging sparse solutions that depend only on a small number of features. In practice, the choice of regularization functions is another modeling decision that can be optimized for using cross-validation, since most common implementations of linear models support a range of penalty functions [44, 92].

5.2.2 Multi-Class Linear Models

The binary formulations in Sect. 5.2.1 can be naturally extended to the *multi-class* setting, where $\mathscr{Y} = \{1, 2, \ldots, C\}$, so that each example is categorized into exactly one of the C distinct classes. Note that this is distinct from the similarly named *multi-label* setting, where each example can be assigned to multiple, non-disjoint classes. While the multi-label setting is often a natural fit for practical applications, it can be handled directly by using C independent binary classifiers.[4]

A natural extension of binary logistic regression can be obtained by defining $\theta = (w_c, b_c)_{c=1}^{C}$, so that each class has its own set of parameters (w_c, b_c).

[3]SVM scores can be converted into probabilities via Platt scaling [94] or isotonic regression [122], but these methods require additional modeling and calibration.

[4]The notion of independence for multi-label problems will be treated more thoroughly when we develop deep learning models.

The probability that a given input x belongs to category j is then defined as

$$\mathbf{P}_\theta[y = j \,|\, x] := \frac{\mathrm{e}^{\langle w_j, x\rangle + b_j}}{\sum_c \mathrm{e}^{\langle w_c, x\rangle + b_c}}. \tag{5.15}$$

Taking the negative log-likelihood of the observed training data results in the following multi-class objective function:

$$f(x, y \,|\, \theta) := -\langle w_y, x\rangle - b_y + \log\left(\sum_c \mathrm{e}^{\langle w_c, x\rangle + b_c}\right). \tag{5.16}$$

Similarly, the linear hinge loss can be generalized by comparing the discriminant score of the true label y for training point x to all other labels c [33]:

$$\begin{aligned} f(x, y \,|\, \theta) &:= \max\left(0, 1 - \langle w_y, x\rangle - b_y + \max_{c \neq y} \langle w_c, x\rangle + b_c\right) \\ &= -\langle w_y, x\rangle - b_y + \max_c \ \ell(y,\, c) + \langle w_c, x\rangle + b_c. \end{aligned} \tag{5.17}$$

Practically speaking, both objectives lead to the same prediction rule:

$$h(x \,|\, \theta) := \operatorname*{argmax}_y \langle w_y, x\rangle + b_y, \tag{5.18}$$

that is, take the label with the highest score.

In multi-class problems, the regularization function is typically applied independently to each w_c and summed: $g(\theta) := \sum_c g_w(w_c)$.

5.2.3 Non-linear Discriminative Models

This section focused on linear models, primarily due to their simplicity, adaptability, and ease of integration with methods discussed in the remainder of this chapter. However, there are a wide range of effective, non-linear discriminative models available to practitioners, which we will briefly describe here. Interested readers are referred to [48] for thorough introductions to these methods.

Most closely related to the linear models described above are *kernel methods* [107]. These methods can be seen as implementing linear models after a non-linear transformation of the data encoded by a kernel function $k(x_1, x_2)$ which generalizes the notion of linear inner product $\langle x_1, x_2\rangle$. Common choices of kernel functions include the *radial basis function* or *Gaussian kernel*:

$$k_\alpha(x_1, x_2) := \exp\left\{-\alpha \|x_1 - x_2\|^2\right\} \tag{5.19}$$

with bandwidth $\alpha > 0$, or the *polynomial kernel*:

$$k_{b,p}(x_1, x_2) := (b + \langle x_1, x_2 \rangle)^p \tag{5.20}$$

with degree $p \geq 1$ and constant $b \geq 0$. Kernel formulations are available for a broad class of suitably regularized models, including the SVM and ℓ_2-regularized logistic regression [103].

Nearest neighbor classifiers [35, 47] operate by querying the training set $\mathscr{X}$ for the nearest examples to a test example x, and predicting $h(x)$ as the majority vote of labels within the nearest neighbor set. This approach is simple to implement, and readily adapts to high-cardinality output label sets. The accuracy of nearest neighbor classifiers depends on the choice of distance function used to determine proximity, and there are a variety of methods available to optimize the metric from a labeled training set [9].

Finally, decision trees [22] operate by recursively partitioning the training set by applying a threshold to individual features. For example, the rule $x_3 \geq 0.75$ would send all examples with the third coordinate less than 0.75 to the left sub-tree, and all others to the right. Recursively applying these rules produces a tree structure, where each leaf of the tree is labeled according to the majority vote of training data that maps to that leaf. Test examples are then classified by the label of the leaf into which they map. Although decision trees are known to be prone to over-fitting, *random forests* [21] ameliorate this by combining the outputs of multiple trees to produce the final classifier. By generating an ensemble of trees from different (random) subsets of the training set and random subsets of features, a random forest tends to be much more robust than a single decision tree, and the general method is highly effective in practice.

5.3 Generative Models

The models in the previous section were discriminative, in the sense that they only need to describe the boundaries between categories, and not the distribution of data within each category. By contrast, *generative models* seek to approximate the data generating process itself by modeling the joint distribution $\mathbf{P}_\theta[x, y]$, rather than the conditional distribution $\mathbf{P}_\theta[y \mid x]$.

Before getting into specific examples of generative models, we will first cast the modeling process into the regularized optimization framework described at the beginning of this chapter, and provide a general overview of statistical inference and parameter estimation.

5.3.1 *Maximum Likelihood Estimation*

When building a generative model, the primary goal is to describe the space of observable data. Consequently, we should strive to make the model distribution $\mathbf{P}_\theta$ match the unknown data distribution $\mathscr{D}$, and our notion of *loss* is tied not to the accuracy of the resulting classifier, but to the *dissimilarity* between $\mathbf{P}_\theta$ and $\mathbf{P}_\mathscr{D}$. From information theory, a particularly useful notion of dissimilarity between probability distributions is the *Kullback–Leibler (KL) divergence* [31, 77]:

$$\mathrm{KL}\left(\mathbf{P}_\mathscr{D} \| \mathbf{P}_\theta\right) := \int_z \log\left(\frac{\mathbf{P}_\mathscr{D}[z]}{\mathbf{P}_\theta[z]}\right) \mathbf{P}_\mathscr{D}[z]\mathrm{d}z. \tag{5.21}$$

which measures the amount of information lost when using distribution $\mathbf{P}_\theta$ to approximate $\mathbf{P}_\mathscr{D}$: the more similar the two distributions are, the smaller the KL-divergence will be.

When $\mathscr{D}$ is fixed, minimizing (5.21) over the choice of θ is equivalent to minimizing the cross-entropy between $\mathbf{P}_\mathscr{D}$ and $\mathbf{P}_\theta$:

$$\underset{\theta}{\mathrm{argmin}}\ \mathrm{KL}\left(\mathbf{P}_\mathscr{D} \| \mathbf{P}_\theta\right) = \underset{\theta}{\mathrm{argmin}} - \int_z \log\left(\mathbf{P}_\theta[z]\right) \mathbf{P}_\mathscr{D}[z]\mathrm{d}z. \tag{5.22}$$

When $\mathscr{D}$ is unknown, except through an I.I.D. sample $\{z_i\}_{i=1}^n \sim \mathscr{D}$, we can approximate (5.22) by the empirical average log-likelihood:

$$-\int_z \log\left(\mathbf{P}_\theta(z)\right) \mathbf{P}_\mathscr{D}[z]\mathrm{d}z = -\mathbf{E}_\mathscr{D}\left[\log \mathbf{P}_\theta[z]\right] \approx -\frac{1}{n}\sum_{i=1}^n \log \mathbf{P}_\theta[z_i]. \tag{5.23}$$

This leads to the standard formulation of *maximum likelihood* parameter estimation: maximizing the probability of $\mathbf{P}_\theta$ generating the training data observed is approximately equivalent to minimizing the KL-divergence between $\mathbf{P}_\theta$ and $\mathbf{P}_\mathscr{D}$. For labeled observations $z = (x, y)$, the corresponding objective function f is then the negative log-likelihood given the model parameters θ:

$$f(x, y \mid \theta) = -\log \mathbf{P}_\theta[x, y]. \tag{5.24}$$

Once θ has been estimated, the prediction rule for an input example x then takes the form:

$$h(x \mid \theta) := \underset{y}{\mathrm{argmax}}\, \mathbf{P}_\theta[x, y]. \tag{5.25}$$

5.3.2 *Bayesian Estimation: Maximum A Posteriori*

In Sect. 5.3.1, there was no explicit mechanism to specify a preference for certain configurations of θ over others (aside from how well it approximates $\mathscr{D}$). The Bayesian approach to resolve this issue is to treat θ as a random variable, alongside the observable data (x, y). In this view, the model probability distribution $\mathbf{P}_\theta$ can be interpreted as conditional on a specific value of θ:

$$\mathbf{P}_\theta[x, y] := \mathbf{P}[x, y \mid \theta], \tag{5.26}$$

and we suppose a *prior distribution* $\mathbf{P}[\theta]$ to express preference for some values of θ over others. Similarly, the corresponding prediction rule for a given value of θ becomes

$$h(x \mid \theta) := \operatorname*{argmax}_y \mathbf{P}[x, y \mid \theta]. \tag{5.27}$$

Bayesian inference consists of computing the *posterior distribution* $\mathbf{P}[\theta \mid S]$ after observing samples $S = \{(x_i, y_i)\}_{i=1}^n \sim \mathscr{D}$ by using Bayes' rule:

$$\mathbf{P}[\theta \mid S] = \frac{\mathbf{P}[S \mid \theta] \times \mathbf{P}[\theta]}{\mathbf{P}[S]}, \tag{5.28}$$

where $\mathbf{P}[S \mid \theta] = \prod_{i=1}^n \mathbf{P}[x_i, y_i \mid \theta]$ factorizes because S is assumed to be drawn I.I.D. Computing (5.28) is difficult because the denominator $\mathbf{P}[S]$ is an unknown quantity (i.e., $\mathscr{D}$) that is generally difficult to estimate. However, if we are only interested in finding a single value of θ which maximizes (5.28), then the $\mathbf{P}[S]$ factor may be safely ignored, since it is constant with respect to the choice of θ. This leads to the *maximum a posteriori* (MAP) formulation of parameter estimation:

$$\operatorname*{argmin}_\theta -\frac{1}{n}\sum_{i=1}^n \log \mathbf{P}[x_i, y_i \mid \theta] - \frac{1}{n}\log \mathbf{P}[\theta]. \tag{5.29}$$

This is derived by taking the logarithm of (5.28), which is equivalent to maximum likelihood inference (5.23), but with an additive term $g(\theta) := -\frac{1}{n}\log \mathbf{P}[\theta]$.[5] MAP inference can thus be viewed as a special case of the generic regularized learning objective (5.6).

The choice of prior distribution $\mathbf{P}[\theta]$ is of utmost importance, and generally depends on several contributing factors such as model structure, existing domain knowledge, and computational convenience. In the following sections, we will discuss the choice of priors for specific models.

5.3.3 Aside: Fully Bayesian Inference

The MAP estimation approach described in the previous section results in classifiers that depend on a single value of θ. If the posterior distribution $\mathbf{P}[\theta \mid S]$ is not strongly

[5]The factor of $1/n$ is not strictly necessary here, but are included for consistency with (5.6).

peaked, or has multiple modes which result in disagreeing predictors, then MAP estimation can become unstable. These situations call for *fully Bayesian inference*, where the uncertainty in θ is explicitly accounted for when making predictions.

Instead of (5.27), a fully Bayesian predictor would marginalize θ out of the joint distribution $\mathbf{P}[x, y, \theta]$ to find the most likely label:

$$h(x) := \operatorname*{argmax}_y \mathbf{P}[x, y] = \operatorname*{argmax}_y \int_\theta \mathbf{P}[x, y \mid \theta] \times \mathbf{P}[\theta] \mathrm{d}\theta. \tag{5.30}$$

In general, this marginal likelihood calculation does not have a closed-form solution, and it can therefore be difficult to compute exactly. When fully Bayesian inference is necessary, it is typically performed by sampling methods such as Markov chain Monte Carlo (MCMC) [62, 87], which can estimate $\mathbf{P}[x, y]$ by drawing samples $\theta \sim \mathbf{P}[\theta]$ and averaging the likelihood estimates $\mathbf{P}[x, y \mid \theta]$. Once the posterior distribution (5.28) has been computed from a training set S, (5.30) can be approximated by sampling from the posterior $\mathbf{P}[\theta \mid S]$ rather than the prior $\mathbf{P}[\theta]$.

There is a rich literature on sampling methods for marginal likelihood, and these methods lie outside the scope of this text [4, 50]. For the remainder of this chapter, we will stick primarily with MAP inference for probabilistic models.

5.3.4 Gaussian Mixture Models

A *Gaussian mixture model* (GMM) consists of a weighted mixture of K multivariate Gaussian distributions [91]. Formally, $\theta = (\omega_k, \mu_k, \Sigma_k)_{k=1}^K$ where ω_k are non-negative weights which sum to 1, and $\mu_k \in \mathbb{R}^d$ and $\Sigma_k \in \mathbb{S}_{++}^d$ denote the mean vector and covariance matrix of the kth mixture component.[6] The probability density at point x is then defined as:

$$\begin{aligned} \mathbf{P}_\theta[x] &:= \sum_k \omega_k \times \mathcal{N}(\mu_k, \Sigma_k) \\ &= \sum_k \omega_k \times |2\pi \Sigma_k|^{-1/2} \times \mathrm{e}^{-\frac{1}{2}\|x-\mu_k\|^2_{\Sigma_k}}, \end{aligned} \tag{5.31}$$

where $\|z\|^2_\Sigma := z^\mathsf{T} \Sigma^{-1} z$. Given a sample $(x_i)_{i=1}^n$, the parameters θ can be inferred by a variety of different strategies, but the most common method is expectation-maximization (EM) [36].

[6] $\mathbb{S}_{++}^d$ denotes the set of $d \times d$ positive definite matrices: Hermitian matrices with strictly positive eigenvalues.

5.3.4.1 Classification with GMMs

Note that (5.31) does not involve the labels y, and can therefore be considered an *unsupervised* model of the data. This can be extended to a multi-class supervised model by fitting a separate GMM $P_{\theta_y}[x|y]$ for each category y. The objective then becomes to find GMM parameters $\theta = (\theta_y, p_y)$, where θ_y contains the parameters of the GMM corresponding to class y, and p_y models the probability of class y. Given an unlabeled example x, the label is predicted as

$$h(x \mid \theta) := \operatorname*{argmax}_y \mathbf{P}_\theta[y \mid x] = \operatorname*{argmax}_y \mathbf{P}_\theta[x \mid y] \times \mathbf{P}_\theta[y], \tag{5.32}$$

where the latter equality follows from Bayes' rule:

$$\mathbf{P}_\theta[y \mid x] = \frac{\mathbf{P}_\theta[x \mid y] \times \mathbf{P}_\theta[y]}{\mathbf{P}_\theta[x]} \propto \mathbf{P}_\theta[x \mid y] \times \mathbf{P}_\theta[y] \tag{5.33}$$

because $\mathbf{P}_\theta[x]$ is (an unknown) constant when searching over y for a given x. The interpretation of (5.32) is similar to that of the multi-class linear models of the previous section: the predicted label is that for which the corresponding generative model assigns highest probability to the input x.

5.3.4.2 Simplifications

There are a few commonly used simplifications to the GMM described in (5.31), as illustrated in Fig. 5.3. The first simplification is to restrict the parameter space so that each Σ_k is a diagonal matrix. This reduces the number of parameters in the model, and simplifies the matrix inverse and determinant calculations in (5.31). This restriction prohibits the model from capturing correlations between variables. However, if the training data has already been decorrelated by a pre-processing step (such as principal components analysis), the diagonal restriction may perform well in practice.

Spherical covariance constraints force $\Sigma_k = \sigma_k I_k$, so that each component has equal variance along each dimension, but that variance can differ from one component to the next. An even more extreme constraint is to force all Σ_k to equal the identity matrix. This restriction, known as *isotropic covariance*, eliminates all variance parameters from the model, so all that are left are the mixture coefficients ω_k and the means μ_k. The spherical restriction may be justified if in addition to being decorrelated, the data are pre-processed to have unit variance along each coordinate, and variance is expected to be independent of component membership. In this case, the GMM can be interpreted as a soft-assignment variant of the K-means clustering algorithm [82].

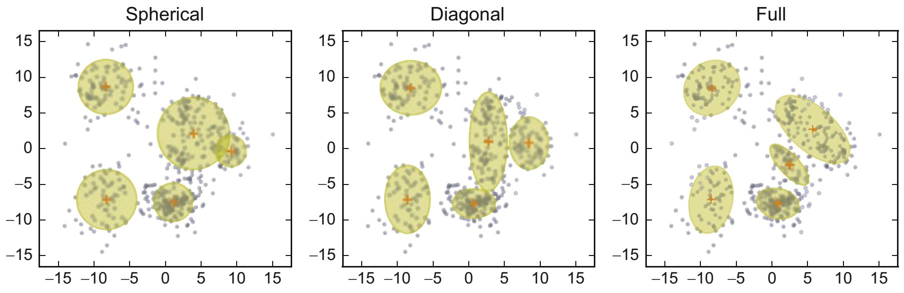

Fig. 5.3 Gaussian mixture models with different covariance constraints applied to the same data set (*blue*). Component means μ_k are indicated by *red plus markers*, and the covariance structures are indicated by *yellow ellipses* covering ±3 standard deviations from μ_k

5.3.4.3 Aside: Maximum Likelihood, or MAP?

As described in the introduction to this section, we have a choice between classical (maximum likelihood) and Bayesian (MAP) inference when estimating the parameters θ of a generative model. For the GMM as described above in (5.31), it should be noted that the classical approach has certain degeneracies that can be avoided by the Bayesian approach.

Specifically, given a training sample, it is possible to make the likelihood arbitrarily large by selecting one component (μ_k, Σ_k) and setting $\mu_k = x_i$ (for some training point x_i), and letting $\Sigma_k = \lambda I$ for some arbitrarily small value $\lambda > 0$ so that the determinant $|\Sigma_k|$ approaches 0. Although it may be rare to encounter this degenerate case in practice, it is possible—especially when the training sample contains outliers (examples far in feature space from most of the training samples). Similar degeneracies can occur when modes of the data lie close to a low-rank subspace. This suggests that maximum likelihood inference may not be the most appropriate choice for estimating GMM parameters.

This situation can be avoided by incorporating prior distributions on the model parameters ω, μ_k, Σ_k which assign low probability to known degenerate configurations. The choice of prior distributions should be guided by domain knowledge and some conceptual understanding of the data, so any general-purpose recipes should be taken as suggestions and treated with skepticism. That said, for computational reasons, it is often preferable to use *conjugate priors*, which can lead to simple parameter updates and computationally efficient learning algorithms.[7]

[7]A probability distribution $\mathbf{P}[\theta]$ is a conjugate prior if the posterior $\mathbf{P}[\theta \mid S]$ has the same form as the prior $\mathbf{P}[\theta]$ [97].

In the case of the GMM, there are three prior distributions in need of definition: $\mathbf{P}[\omega]$, $\mathbf{P}[\Sigma]$, and $\mathbf{P}[\mu]$. Because ω is a categorical random variable, the (symmetric) Dirichlet distribution can be used since it is conjugate to the categorical distribution:

$$\mathbf{P}_\alpha[\omega] := \frac{\Gamma(\alpha K)}{\Gamma(\alpha)^K} \prod_{k=1}^{K} \omega_k^{\alpha-1}. \tag{5.34}$$

The $\alpha > 0$ variable is a hyper-parameter: for $\alpha \geq 1$, ω tends to be dense; for $\alpha < 1$, ω tends to concentrate on a few components, which can be used to effectively eliminate unused components.

The covariance prior $\mathbf{P}[\Sigma]$ can be somewhat more difficult to define. For diagonal covariance models, it is common to factor the prior over each variance component $\mathbf{P}[\Sigma] = \mathbf{P}[\sigma_i^2]$, and use a prior with support limited to positive reals, such as the log-normal or gamma distributions. If the prior assigns 0 probability to $\sigma_i^2 = 0$—as the log-normal distribution does, or gamma with shape parameter $\alpha > 1$—then the degenerate cluster issue described above can be effectively prevented. For full-covariance models, the Wishart distribution (a multivariate extension of the gamma distribution with support over positive definite matrices) can be used to achieve similar results. Each of these prior distributions has additional hyper-parameters which specify the location and dispersion of probability mass.

Finally, the prior on cluster means $\mathbf{P}[\mu]$ is often taken to be a standard, multivariate Gaussian when $\mathscr{X} = \mathbb{R}^d$. However, if additional constraints are known, e.g., observations are non-negative magnitude spectra so $\mathscr{X} = \mathbb{R}_+^d$, then a coordinate-wise log-normal or gamma distribution might be more appropriate. For a more thorough discussion of priors for generative models, we refer interested readers to [85, Chap. 5].

5.3.4.4 Parameter Estimation

While it is relatively straightforward to implement the expectation-maximization (EM) algorithm for the maximum likelihood formulation of a GMM, the task can become significantly more complex in the MAP scenario, as the update equations may no longer have convenient, closed-form solutions. Variational approximate inference methods are often used to simplify the parameter estimation problem in this kind of setting, usually by computing the MAP solution under a surrogate distribution with a more computationally convenient factorization [118]. A proper treatment of variational inference is beyond the scope of this text, and it should be noted that although the resulting algorithms are "simpler" (more efficient), the derivations can be more complex as well. Software packages such as Stan [23] and Edward [114] can ease the burden of implementing variational inference by automating much of the tedious calculations.

5.3.4.5 How Many Components?

Throughout this section, we have assumed that the number of mixture components K was fixed to some known value. In practice, however, the best K is never known a priori, so practitioners must find some way to select K. There are essentially three data-driven approaches to selecting K: information criteria, Dirichlet process mixtures, and utility optimization.

The first approach comes in a wide range of flavors: Akaike's information criterion (AIC) [2], Bayesian information criterion (BIC) [105], or widely applicable information criterion (WAIC) [119]. The common thread throughout these methods is that one first constructs a set of models—for a GMM, each model would correspond to a different choice of K—and select the one which best balances accuracy (likelihood of observed data) against model complexity. The methods differ in how "model complexity" is estimated, and we refer interested readers to Watanabe [119] for a survey of the topic.

The second approach, Dirichlet process mixtures [5], implicitly supports a countably infinite number of mixture components, and estimates K as another model parameter along with mixing weights, means, and variances [15, 98]. This model can be approximated by setting K to some reasonable upper limit on the acceptable number of components, imposing a sparse Dirichlet prior ($\alpha < 1$) over ω, and estimating the parameters just as in the case where K is fixed [69]. Then, any components with sufficiently small mixture weights ω_i (e.g., those whose combined weight is less than 0.01) can be discarded with negligible impact on the corresponding mixture density.

Finally, utility-based approaches select the model which works best for a given application, i.e., maximizes some expected utility function. In classification problems, the natural utility function to use would be classification accuracy of the resulting predictor (5.32). Concretely, this would amount to treating K as another hyper-parameter to be optimized using cross-validation, with classification accuracy as the selection criterion.

5.3.5 Hidden Markov Models

So far in this chapter, the models described have not directly addressed temporal dynamics of sound. To model dynamics, we will need to treat a sequence of feature observations as a single object, which we will denote by $x = (x[1], x[2], \ldots, x[T])$.[8]

[8]Note that although we use T to denote the length of an arbitrary sequence x, it is not required that all sequences have the same length.

In general, the likelihood of a sequence observation $\mathbf{P}_\theta[x]$ can be factored as follows:

$$\begin{aligned}\mathbf{P}_\theta[x] &= \mathbf{P}_\theta\left[x[1], x[2], \ldots, x[T]\right] \\ &= \mathbf{P}_\theta\left[x[1]\right] \times \prod_{t=1}^{T-1} \mathbf{P}_\theta\left[x[t+1] \mid x[1], x[2], \ldots, x[t]\right].\end{aligned} \tag{5.35}$$

The *Markov* assumption asserts that this distribution factors further:

$$\mathbf{P}_\theta[x] := \mathbf{P}_\theta\left[x[1]\right] \times \prod_{t=1}^{T-1} \mathbf{P}_\theta\left[x[t+1] \mid x[t]\right]. \tag{5.36}$$

That is, the distribution at time $t+1$ conditional on time t is independent of any previous time $t' < t$. A *hidden Markov model* (HMM) asserts that all dynamics are governed by hidden discrete "state" variables $z[t] \in \{1, 2, \ldots, K\}$, and that an observation $x[t]$ at time t depends only on the hidden state $z[t]$ [7, 96].

Formally, an HMM is defined by a joint distribution of the form:

$$\mathbf{P}_\theta[x, z] := \prod_{t=1}^{T} \mathbf{P}_\theta\left[z[t] \mid z[t-1]\right] \times \mathbf{P}_\theta\left[x[t] \mid z[t]\right]. \tag{5.37}$$

There are three distinct components to (5.37):

- $\mathbf{P}_\theta\left[z[1] \mid z[0]\right]$ is the *initial state model*, which determines the probability of starting a sequence in each state[9];
- $\mathbf{P}_\theta\left[z[t] \mid z[t-1]\right]$ is the *transition model*, which governs how one hidden state transitions to the next; and
- $\mathbf{P}_\theta\left[x[t] \mid z[t]\right]$ is the *emission model*, which governs how each hidden state generates observed data.

If there are K possible values for a hidden state, then the transition model can be defined as a collection of K categorical distributions over the K hidden states. The parameters of these distributions are often collected into a $K \times K$ stochastic matrix known as the *transition matrix* V, where

$$V_{ij} = \mathbf{P}_\theta\left[z[t] = i \mid z[t-1] = j\right]. \tag{5.38}$$

Similarly, the initial state model can also be defined as a categorical distribution over the K hidden states.

[9]For ease of notation, we denote the initial state distribution as $\mathbf{P}_\theta\left[z[1] \mid z[0]\right]$, rather than the unconditional form $\mathbf{P}_\theta\left[z[1]\right]$.

The definition of the emission model depends on the form of the observed data. A common choice, when $x[t] \in \mathbb{R}^d$ is to define a multivariate Gaussian emission model for each hidden state:

$$\mathbf{P}_\theta\left[x \mid z[t] = k\right] := \mathcal{N}(\mu_k, \Sigma_k). \tag{5.39}$$

This model is commonly referred to as the Gaussian-HMM. Note that the specification of the emission model does not depend on the transition model, and any reasonable emission model may be used instead. Emission models can themselves also be mixture models, and GMMs are particularly common.

Once the parameters of the model have been estimated (Sect. 5.3.5.2), the most likely hidden state sequence z for an observed sequence x can be inferred by the Viterbi algorithm [117]. The resulting state sequence can be used to segment the sequence into contiguous subsequences drawn from the same state. In audio applications, this can correspond directly to the temporal activity of a class or sound source [64].

5.3.5.1 Discriminative HMMs

The most common way to apply HMMs for classification is to impose some known structure over the hidden state space. For example, in speech recognition applications, we may prefer a model where each hidden state corresponds to a known phoneme [73]. If labeled training data is available, where each observation sequence $x = (x[1], x[2], \ldots, x[T])$ has a corresponding label sequence $y = (y[1], y[2], \ldots, y[T])$, then we can directly relate the hidden state space to the label space $\mathcal{Y}$. While one could use a discriminative model to independently map each observation to a label, this would ignore the temporal dynamics of the problem. Integrating the classification with an HMM can be seen as a way of imposing temporal dynamics over model predictions.

Recall that there are three quantities to be estimated in an HMM: the initial state distribution, the state transition distribution, and the emission distribution. When labeled training data is available—i.e., the state variable for each observation is also observed—the first two distributions can be estimated directly from the labels, since they are conditionally independent of the input data x given the state. In practice, this amounts to estimating the parameters of $K + 1$ categorical distributions—K for the transition distributions and one for the initial state distribution—from the observed labeled sequences.

All that remains is to characterize the emission distributions. This can be done by applying Bayes' rule to the observation model, now using y to indicate states instead of z:

$$\mathbf{P}_\theta\left[x[t] \mid y[t] = k\right] = \frac{\mathbf{P}_\theta\left[y[t] = k \mid x[t]\right] \times \mathbf{P}_\theta\left[x[t]\right]}{\mathbf{P}_\theta\left[y[t] = k\right]} \tag{5.40}$$

which expresses the emission probability in terms of the conditional class likelihood $\mathbf{P}_\theta\,[y[t] = k \mid x[t]]$, the marginal probability of the class occurring $\mathbf{P}_\theta\,[y[t] = k]$, and the marginal probability of the observation $\mathbf{P}_\theta\,[x[t]]$. The conditional class likelihood can be estimated by any probabilistic discriminative classifier, e.g., logistic regression (Sect. 5.2.1.2) or a multi-layer perceptron (Sect. 5.4.2). The marginal probability of the class is a categorical distribution that can be estimated according to the statistics of the labeled training data.

Finally, the marginal probability of the observation $\mathbf{P}_\theta\,[x[t]]$ is generally difficult to estimate, but luckily it is not often needed. Recall that the practical application of the HMM is to produce a sequence of labels $y = (y[1], y[2], \ldots, y[T])$ from an unlabeled observation sequence $x = (x[1], x[2], \ldots, x[T])$ following the prediction rule:

$$h(x \mid \theta) := \operatorname*{argmax}_y \mathbf{P}_\theta[x, y]. \tag{5.41}$$

Substituting (5.40) and (5.37) into (5.41) yields

$$\begin{aligned}
\mathbf{P}_\theta\,[x, y] &= \prod_{t=1}^{T} \mathbf{P}_\theta\,[y[t] \mid y[t-1]] \times \mathbf{P}_\theta\,[x[t] \mid y[t]] && \text{(5.42a)}\\
&= \prod_{t=1}^{T} \mathbf{P}_\theta\,[y[t] \mid y[t-1]] \times \mathbf{P}_\theta\,[x[t]] \times \frac{\mathbf{P}_\theta\,[y[t] \mid x[t]]}{\mathbf{P}_\theta\,[y[t]]} && \text{(5.42b)}\\
&= \left(\prod_{t=1}^{T} \mathbf{P}_\theta\,[y[t] \mid y[t-1]] \times \frac{\mathbf{P}_\theta\,[y[t] \mid x[t]]}{\mathbf{P}_\theta\,[y[t]]}\right) \times \prod_{t=1}^{T} \mathbf{P}_\theta\,[x[t]] && \text{(5.42c)}\\
&\propto \prod_{t=1}^{T} \mathbf{P}_\theta\,[y[t] \mid y[t-1]] \times \frac{\mathbf{P}_\theta\,[y[t] \mid x[t]]}{\mathbf{P}_\theta\,[y[t]]}, && \text{(5.42d)}
\end{aligned}$$

where the $\mathbf{P}_\theta\,[x[t]]$ factors can be ignored since they do not affect the maximization over choice of y. Consequently, the sequence prediction (5.41) can be computed by the Viterbi algorithm using only the discriminative classifier's point-wise output and the empirical unigram- and bigram-statistics, $\mathbf{P}_\theta\,[y[t]]$ and $\mathbf{P}_\theta\,[y[t] \mid y[t-1]]$, of observed label sequences.

In addition to attaching specific meaning to the "hidden" state variables (i.e., correspondence with class labels), there are two computational benefits to this approach. First, it can be applied to any probabilistic classifier, and it is often used as a post-processing technique to reduce errors resulting from frame-wise classifiers. Second, discriminative classifiers are often easier to train than generative models, since they typically require less observation data, and the resulting models tend to be more accurate in practice.

The discriminative HMM approach described here can be viewed as a special case of a conditional random field (CRF) [78], where the model parameters have been estimated independently. A more general CRF-based approach would jointly estimate all model parameters, which can improve accuracy in practice.

5.3.5.2 Priors and Parameter Estimation

Parameter estimation for HMMs can be done either in maximum likelihood or MAP formulations, and the resulting algorithms are qualitatively similar to the case for Gaussian mixture models.[10] In particular, a Bayesian formulation of the HMM looks nearly identical to that of the GMM—with the initial state model $\mathbf{P}[z[1] \mid z[0], \theta]$ acting in place of the mixture weights $\mathbf{P}[\omega \mid \theta]$—and the only additional set of parameters in need of a prior is the transition matrix. Since each row $V_{\cdot,j}$ of the transition matrix is a categorical distribution, it is again natural to impose a Dirichlet prior over each row of V. For details on Bayesian HMM inference, we refer readers to Beal [8].

Just as in the GMM case, the number of hidden states K is another hyper-parameter to be estimated, and it can be done via any of the methods described in Sect. 5.3.4.5. In the discriminative case where hidden states are matched to observable labels, this issue does not arise.

5.4 Deep Models

In this section, we provide a brief overview of the so-called *deep learning* architectures. While the term *deep learning* can apply to a wide range of different types of models, we will focus specifically on discriminative classification models which include a non-linear transformation of input data that is jointly optimized with the classifier. For a more thorough introduction, we refer interested readers to Goodfellow et al. [54].

5.4.1 Notation

Deep models are often characterized by compositions of non-linear functions. For brevity, we will denote the sequential composition of k functions by the $\circledcirc$ symbol, defined as:

$$\left(\bigcirc\!\!\!\!\circ_{i=1}^{m} f_i \right)(x) := (f_m \circ f_{m-1} \circ \cdots \circ f_1)(x). \tag{5.43}$$

Each f_i should be interpreted as a stage or layer of processing, which transforms the output of f_{i-1} to the input of f_{i+1}.

[10]The well-known Baum–Welch algorithm for HMM parameter estimation is a special case of expectation-maximization [96].

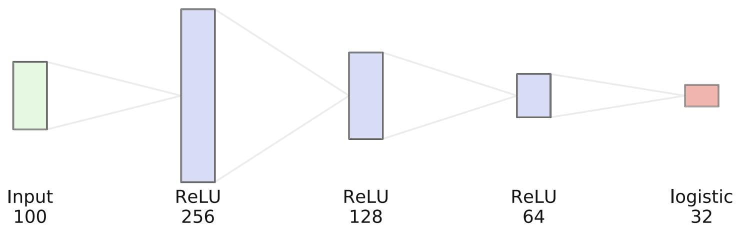

Fig. 5.4 An example illustration of a multi-layer perceptron (MLP) with four layers. The input $x \in \mathbb{R}^{100}$ is mapped through three intermediate layers with $d_1 = 256$, $d_2 = 128$, and $d_3 = 64$ and rectified linear unit (ReLU) transfer functions. The output layer in this example maps to 32 independent classes with a logistic transfer function

5.4.2 Multi-Layer Perceptrons

The simplest, and oldest family of "deep" models is the multi-layer perceptron (MLP) [99, 101]. As illustrated in Fig. 5.4, an MLP is defined by a sequence of *layers*, each of which is an affine transformation followed by a non-linear *transfer function* ρ:

$$f_i(z \mid \theta) := \rho_i \left(w_i^\mathsf{T} z + b_i \right), \tag{5.44}$$

where the parameters $\theta = (w_i, b_i, \rho_i)_{i=1}^m$ have weights $w_i \in \mathbb{R}^{d_{i-1} \times d_i}$, biases $b_i \in \mathbb{R}^{d_i}$, and transfer functions $\rho_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$. Each layer maps data from $\mathbb{R}^{d_{i-1}}$ to $\mathbb{R}^{d_i}$, which dictates the shape of the resulting MLP.[11] For input data $x \in \mathbb{R}^d$, we define $d_0 := d$, and for categorical prediction tasks, we define $d_m := |\mathscr{Y}|$ as the number of labels.

The final (output) layer f_m of an MLP is typically defined as a linear model in one of the forms described in Sect. 5.2. The "internal" layers $f_1 \ldots f_{m-1}$ can be interpreted as a "feature extractor." One motivation for this architecture is that by jointly optimizing the internal and output layers, the model finds a feature extractor which separates the data so that the output layer performs well. In contrast to the linear models described in Sect. 5.2, this approach directly benefits from multi-class and multi-label data because it can leverage observations from all classes in constructing the shared representation.

The surrogate loss f is defined in terms of the output of the final layer:

$$f(x, y \mid \theta) := f_{\text{err}} \left(\left(\bigodot_{i=1}^m f_i \right) (x),\ y \right), \tag{5.45}$$

[11] Some authors refer to the layer dimension d_i as *width*. This terminology can be confusing when applied to spatio-temporal data as in Sect. 5.4.3, so we will use *dimension* to indicate d_i and retain *width* to describe a spatial or temporal extent of data.

where f_{err} is a standard surrogate loss function as described in Sect. 5.2 that compares the output of the final layer f_m to the target label y. Just as in the previous sections, the prediction rule corresponding to (5.45) is to choose the label which would minimize the objective:

$$h(x \mid \theta) := \operatorname*{argmin}_y f(x, y \mid \theta). \tag{5.46}$$

Typically, this will simplify to an argmax over the output variables (for multi-class problems), or a thresholding operation (for binary problems).

5.4.2.1 Transfer and Objective Functions

The transfer function ρ_i—also known as an *activation function* or *non-linearity*—allows the model to learn non-linear structure in data.[12] As illustrated in Fig. 5.5, there are a variety of commonly used transfer functions.

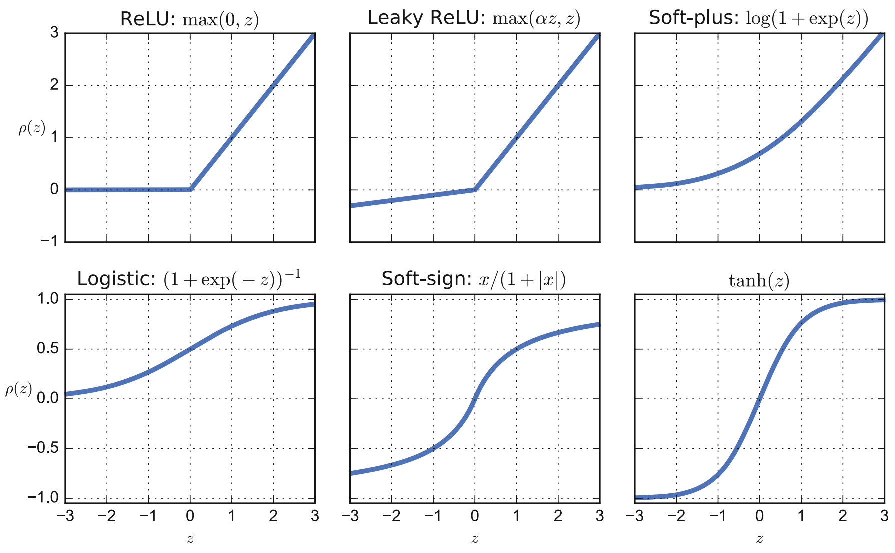

Fig. 5.5 A comparison of various transfer functions ρ. Note that some saturate on both negative and positive inputs (logistic, tanh, soft-sign), while others saturate only on negative inputs (ReLU, soft-plus), or not at all (leaky ReLU)

[12]To see this, observe that if ρ_i is omitted, then the full model $f(x \mid \theta)$ is a composition of affine functions, which is itself an affine function, albeit one with rank constraints imposed by the sequence of layer dimensions.

The choice of ρ_i for internal layers ($i < m$) is generally at the practitioner's discretion. As illustrated in Fig. 5.5, some non-linearities approach saturating values, such as tanh going to ± 1, or logistic going to $\{0, 1\}$ as z diverges from 0. When the non-linearity saturates to a constant, its derivative goes to 0 in that region of its domain, and as a result, the error signal cannot (easily) propagate through to earlier layers. Although techniques exist to limit this behavior (e.g., by scaling the activations to stay within the non-saturating regions [68]), it is simpler in practice to use a one-sided or non-saturating transfer function, such as the *rectified linear unit* (ReLU) [86] or leaky ReLU [81].

Typically the choice of ρ_m (the output layer) is dictated by the structure of the output space. If the output is a multi-label prediction, then the logistic function ($\rho_m = \sigma$) provides a suitable transfer function that can be interpreted as the likelihood of each label given the input data. In this setting, the label is usually encoded as a binary vector $y \in \{0, 1\}^C$ (for C labels), and the standard loss function is the sum of log-likelihoods for each label:

$$f_{\text{err}}(\hat{y}, y) := \sum_{c=1}^{C} -y_c \log \hat{y}_c - (1 - y_c) \log (1 - \hat{y}_c), \tag{5.47}$$

where $\hat{y} = (\bigodot_{i=1}^{m} f_i)(x)$ is the output of the MLP on input x. This loss function is also known as the *binary cross-entropy* loss, since it is equivalent to the sum of cross-entropies between K pairs of Bernoulli random variables.

For multi-class problems, the soft-max function provides a normalized, non-negative output vector that can be interpreted as a categorical probability distribution:

$$\rho_{\text{softmax}}(z)_k := \frac{\exp(z_k)}{\sum_j \exp(z_j)}. \tag{5.48}$$

In multi-class problems, the label y is typically encoded as a binary vector with exactly one non-zero entry. The standard loss function is the categorical cross-entropy:

$$f_{\text{err}}(\hat{y}, y) := -\sum_{c=1}^{C} y_c \log \hat{y}_c. \tag{5.49}$$

5.4.2.2 Initialization

Related to the choice of transfer function is the issue of weight initialization. Because gradients do not propagate when the input to the transfer function lies in its saturating regions, it is beneficial to randomly initialize weights w_i and biases b_i such that $\mathbf{E}_{w_i,b}\left[\rho\left(w_i^{\mathsf{T}} z + b_i\right)\right]$ has non-zero derivative. Glorot and Bengio [52] derive an initialization scheme using weights w_{ij} sampled randomly from the interval $\pm d_{i-1}^{-1/2}$, with the implicit assumption that the input z is bounded in $[-1, 1]$, as is the case when using symmetric, saturating transfer functions (e.g., logistic or tanh).

He et al. [63] argue that this scheme is ill-suited for networks in which the input z has non-zero expectation, as is the case for networks with ReLU activations. Instead, He et al. recommend that weights be initialized as $w_{ij} \sim \mathcal{N}\left(0, \sqrt{2/d_{i-1}}\right)$ for ReLU networks, or more generally,

$$w_{ij} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{(1+\alpha^2)d_{i-1}}}\right) \tag{5.50}$$

for leaky ReLU networks with parameter $\alpha \geq 0$.

We note that most common implementations provide these initialization schemes by default [1, 27, 38], but it is still up to the practitioner to decide which initialization to use in conjunction with the choice of transfer function.

5.4.2.3 Learning and Optimization

In general, the form of f does not lend itself to closed-form solutions, so the parameters are estimated by iterative methods, usually some variation of gradient descent:

$$\theta \mapsto \theta - \eta \nabla_\theta f(x, y \mid \theta) \tag{5.51}$$

where ∇_θ denotes the gradient operator with respect to parameters θ, and $\eta > 0$ is a learning rate that controls how far to move θ from one iteration to the next.

Because f is defined as a composition of simpler functions, the gradient $\nabla_\theta f$ is decomposed into its individual components (e.g., ∇_{w_i} or ∇_{b_j}), which are computed via the chain rule. This process is also known as *back-propagation*, since the calculation can be seen as sending an error signal back from the output layer through the sequence of layers in reverse-order [101]. In the past, calculating the gradients of (5.45) was a tedious, mechanical chore that needed to be performed for each model. However, in recent years, nearly all deep learning frameworks include automatic differentiation tools, which remove this burden of implementation [1, 11, 28, 71].

Due to the computational and memory complexity of computing gradients over a large training set, the common practice is to use *stochastic gradient descent* (SGD) [18], which estimates the gradient direction at each step k by drawing a small *mini-batch* B_k of samples from the training set S, and approximating the gradient:

$$\hat{\nabla}_\theta f := \frac{1}{|B_k|} \sum_{(x,y)\in B_k} \nabla_\theta f(x, y \mid \theta) \tag{5.52a}$$

$$\approx \frac{1}{|S|} \sum_{(x,y)\in S} \nabla_\theta f(x, y \mid \theta) = \nabla_\theta f \tag{5.52b}$$

$$\approx \mathbf{E}_{\mathscr{D}} \left[\nabla_\theta f(x, y \mid \theta)\right]. \tag{5.52c}$$

It is also common to accelerate the standard SGD approach described above by using *momentum* methods [88, 95, 111], which re-use gradient information from previous iterations to accelerate convergence. Similarly, adaptive update schemes like AdaGrad [41] and ADAM [75] reduce the dependence on the step size η, and can dramatically improve the rate of convergence in practice.

Finally, to prevent over-fitting of MLP-based models, it is common to use *early stopping* as a form of regularization [109], rather than minimizing (5.45) over the training set until convergence. This is usually done by periodically saving check-points of the model parameters θ, and then validating each check-point on held-out data as described in Sect. 5.1.2.

5.4.2.4 Discussion: MLP for Audio

Multi-layer perceptrons form the foundation of deep learning architectures, and can be effective across a wide range of domains. However, the MLP presents some specific challenges when used to model audio.

First, and this is common to nearly all models discussed in this chapter, is the choice of input representation. Practitioners generally have a wide array of input representations to choose from—time-domain waveforms, linear-frequency spectrograms, log-frequency spectrograms, etc.—and this choice influences the efficacy of the resulting model. Moreover, the scale of the data matters, as described in Sect. 5.4.2.2. This goes beyond the simple choice of linear or logarithmic amplitude spectrogram scaling: as discussed in the previous section, training is difficult when transfer functions operate in their saturating regions. A good heuristic is to scale the input data such that the first layer's transfer function stays within non-saturating region in expectation over the data. Coordinate-wise standardization (also known as *z-scoring*) using the training set's mean and variance statistics (μ, σ^2) accomplishes this for most choices of transfer functions, since each coordinate maps most of the data to the range [−3, +3][13]:

$$x_i \mapsto \frac{x_i - \mu_i}{\sigma_i}. \tag{5.53}$$

In practice, coordinate-wise standardization after a log-amplitude scaling of spectral magnitudes works well for many audio applications.

[13]Note that batch normalization accomplishes this scaling implicitly by estimating these statistics during training [68].

Second, an MLP architecture requires that all input have a fixed dimension d, which implies that all audio signals must be cropped or padded to a specified duration. This is typically achieved by dividing a long signal (or spectrogram) $x \in \mathbb{R}^{T \times n}$ into small, fixed-duration observations $x_i \in \mathbb{R}^{\delta \times n}$. Observations x_i can be interpreted as vectors of dimension $d = n\delta$, and processed independently by the MLP. In doing so, care must be taken to ensure that the window length δ is sufficiently long to capture the target concept.

Finally, MLPs do not fully exploit the structure of audio data implicit in time or frequency dimensions. For example, if two observations x_1, x_2 are derived from a signal spanning frame indices $[t, t + \delta]$ and $[t + 1, t + \delta + 1]$, respectively, the MLP outputs for $f(x_1)$ and $f(x_2)$ can diverge significantly, even though the inputs differ only by two frames. Consequently, MLPs trained on audio data can be sensitive to the relative positioning of an observation within the window. For non-stationary target concepts, this presents a great difficulty for MLP architectures, since they effectively need to detect the target event at every possible alignment within the window. The remaining sections of this chapter describe methods to circumvent this problem by exploiting the ordering of time or frequency dimensions.

5.4.3 Convolutional Networks

Convolutional networks are explicitly designed to overcome the limitations of MLPs described in the previous Sect. [79]. There are two key ideas behind convolutional networks:

1. statistically meaningful interactions tend to concentrate locally, e.g., within a short time window around an event;
2. shift-invariance (e.g., in time) can be exploited to share weights, thereby reducing the number of parameters in the model.

Convolutional networks are well-suited to applications in which the desired output is a sequence of predictions, e.g., time-varying event detection, and the concepts being modeled derive only from local interactions. In this section, we will describe one-dimensional and two-dimensional convolutional networks. Though the idea generalizes to higher dimensions, these two formulations are the most practically useful in audio applications.

5.4.3.1 One-Dimensional Convolutional Networks

Given an input observation $z \in \mathbb{R}^{T \times d}$, a one-dimensional convolutional filter with coefficients $w \in \mathbb{R}^{n \times d}$ and bias b produces a response $\rho\,(w * z + b)$, where $w * z$

denotes the "valid"[14] discrete convolution of w with z[15]:

$$(w * z)[t] := \sum_{j=1}^{n} \langle w[j],\ z\,[t + j - \lceil n/2 \rceil] \rangle\,. \tag{5.54}$$

Here, $n \leq T$ denotes the size of the receptive field of the filter w, j indexes the filter coefficients and position within the input signal, and d indicates the dimensionality (number of channels) in the input. By convention, the receptive field n is usually chosen to be an odd number so that responses are centered around an observation.

A convolutional layer $f_i : \mathbb{R}^{T_{i-1} \times d_{i-1}} \to \mathbb{R}^{(T_{i-1} - n_i + 1) \times d_i}$ consists of d_i convolutional filters, collectively denoted as w_i, and bias terms (collected as b_i):

$$f_i(z \mid \theta) := \rho_i\,(w_i * z + b_i)\,. \tag{5.55}$$

The output of f_i, sometimes called a *feature map* is of slightly reduced extent than the input (due to valid-mode convolution), and can be interpreted as sliding an MLP with weights $w \in \mathbb{R}^{d_{i-1} \times d_i}$ over every position in the input z. An example of this architecture is illustrated in Fig. 5.6.

Note that although the input and output of a convolutional layer are two-dimensional, the first dimension is assumed to encode "temporal position" (over which the convolution ranges) and the second dimension encodes (unordered) filter channel responses as a function of position. When the input has only a single observation channel (e.g., waveform amplitude), which is represented as $x \in \mathbb{R}^{T_0 \times 1}$. For higher-dimensional input—e.g., spectrograms—d_0 corresponds to the number of observed features at each time step (e.g., number of frequency bins).

Cascading convolutional layers can be interpreted as providing hierarchical representations. However, in the form given above, the receptive field of the ith layer's filter is linear in i. *Pooling layers* down-sample feature maps between convolutional layers, so that deeper layers effectively integrate larger extents of data. A one-dimensional pooling layer has two parameters: width r and stride s:

$$f_i(z \mid \theta)[t] := \operatorname{agg}\left(z[ts + j] \,\middle|\, j \in \left[-\left\lfloor \frac{r}{2} \right\rfloor, \left\lfloor \frac{r}{2} \right\rfloor\right]\right), \tag{5.56}$$

where agg denotes an aggregation operator, such as max() or sum(), and is applied independently to each channel. Max-pooling is particularly common, as it can be interpreted as a softened version of a logical-or, indicating whether filter had positive response anywhere within the window around z_{ts}. Usually, the pooling stride is set to match the width ($s = r$), so that there is minimal redundancy in the output of the pooling layer, and the result is a downsampling by a factor of s.

[14]A *valid-mode* convolution is one in which the response is computed only at positions where the signal z and filter w fully overlap. For $z \in \mathbb{R}^T$ and $w \in \mathbb{R}^n$, the valid convolution $w * z \in \mathbb{R}^{T-n+1}$.

[15]Technically, (5.54) is written as a cross-correlation and not a convolution. However, since the weights w are variables to be learned, and all quantities are real-valued, the distinction is not important.

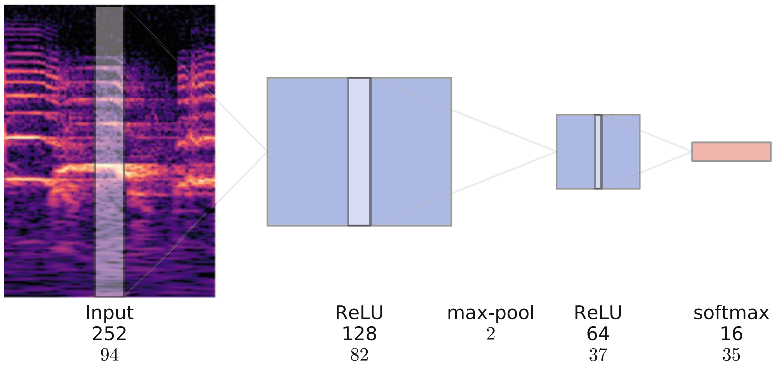

Fig. 5.6 An example of a one-dimensional convolutional network using a spectrogram as input; convolution is performed only over the time dimension (horizontal axis). The vertical axis corresponds to the dimension of each layer. The *shaded regions* indicate the effective receptive field of a single filter in the subsequent layer at the center position. This network takes input in $\mathbb{R}^{T\times 252}$ (in this example, $T = 94$ frames), and applies: $d_1 = 128$ convolutional filters ($n_1 = 13$ frames) with ReLU activation, a downsampling of $r = 2$; $d_3 = 64$ convolutional filters ($n_3 = 5$ frames) with ReLU activation, and finally a convolutional soft-max layer ($n_4 = 3$ frames) mapping to $d_4 = 16$ classes

Typical convolutional architectures alternate convolution layers with pooling layers, which ultimately results in an output layer of shape $T_m \times d_m$, where $T_m < T_0$ is the result of successive pooling and valid-mode convolution operations. Note that T_m is generally a function of T_0, and will differ for inputs of different length. Care should be taken during training to align the sampling rate of labels y to that of the model's output layer f_m, but this is usually a simple task (e.g., downsampling y).

However, when the desired output exists only at the recording level, then some form of aggregation is required so that the output layer's shape conforms to that of the labels. The two most common approaches to reconciling output shapes are

1. use *global* pooling operators across the convolutional dimension—e.g., max over the entire time axis—followed by a standard MLP architecture; or
2. pass the convolutional outputs directly into fully connected MLP layers (Sect. 5.4.5).

Note that for the global pooling approach to work in general, the model must be able to accommodate input of variable length; this is a common enough operation that most software implementations support global pooling operators. The second approach implicitly limits the model to operate on fixed-dimensional inputs, even at test time. This can make the resulting model inconvenient to deploy, since observation windows must be manually extracted and passed through the model, but the models tend to perform well in practice because they preserve temporal relations among feature activations within the observation.

One-dimensional convolutional networks are often applied to spectrogram representations of audio [65, 80, 116]. While this approach is often successful, it is worth noting that it cannot learn filters which are invariant to frequency transposition. As a result, it may require a large number of (redundant) first-layer filters to accurately model phenomena which move in frequency.

5.4.3.2 Two-Dimensional Convolutional Networks

The one-dimensional convolutional method described in the previous section generalizes to higher-dimensional data, where multiple dimensions possess a proper notion of locality and ordering. Two-dimensional convolutional architectures are especially common, due to their natural application in image processing [79], which can in turn be adapted to time-frequency representations of audio. The benefits of two-dimensional convolutional architectures on time-frequency representations include a larger effective observation window (due to temporal framing, as in one-dimensional convolutional networks), the ability to leverage frequency decompositions to more clearly locate structures of interest, and the potential for learning representations which are invariant to frequency transposition.

Technically, two-dimensional convolutional layers look much the same as their one-dimensional counterparts. An input observation is now represented as a three-dimensional array $z \in \mathbb{R}^{T \times U \times d}$ where T and U denote temporal and spatial extents, and d denotes non-convolutional input channels. The filter coefficients similarly form a three-dimensional array $w \in \mathbb{R}^{n \times p \times d}$, and the convolution operator $w * z$ is accordingly generalized:

$$(w * z)[t, u] := \sum_{j=1}^{n} \sum_{k=1}^{p} \langle w[j, k],\ z\left[t + j - \lceil n/2 \rceil,\ u + k - \lceil p/2 \rceil\right] \rangle . \tag{5.57}$$

The corresponding layer $f_i : \mathbb{R}^{T_{i-1} \times U_{i-1} \times d_{i-1}} \to \mathbb{R}^{T_i \times U_i \times d_i}$ otherwise operates analogously to the one-dimensional case (5.55), and pooling operators generalize in a similarly straightforward fashion. For a spectrogram-like input $x \in \mathbb{R}^{T_0 \times U_0 \times d_0}$, we take T_0 to be the number of frames, U_0 the number of frequency bins, and $d_0 = 1$ to indicate the number of channels. Note that larger values of d_0 are possible, if, for instance, one wished to jointly model stereo inputs ($d_0 = 2$, for left and right channels), or some other time- and frequency-synchronous multi-channel representation.

Two-dimensional convolutional networks can be used to learn small, localized filters in the first layer, which can move both vertically (in frequency) and horizontally (in time) [67, 102]. Unlike one-dimensional convolutions, two-dimensional convolution is only well-motivated when the input representation uses a log-scaled frequency representation (e.g., a constant-Q transform), so that the ratio of frequencies covered by a filter of height p bins remains constant regardless of its position.

An example of this architecture is illustrated in Fig. 5.7. The first layer filters in this kind of architecture tend to learn simple, local features, such as transients or sustained tones, which can then be integrated across large frequency ranges by subsequent layers in the network.

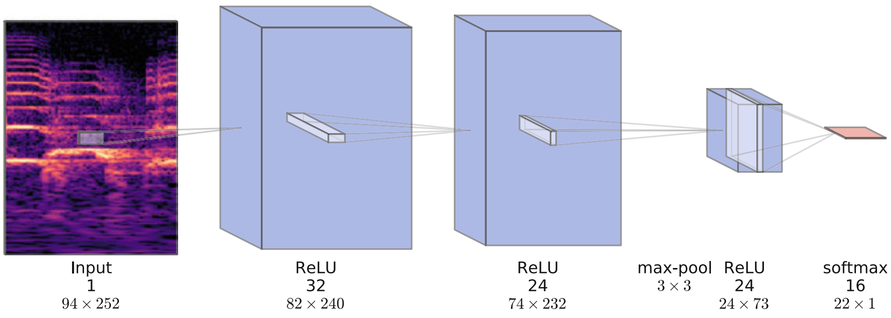

Fig. 5.7 An example of a two-dimensional convolutional network with spectrogram input and local filters. The depth axis corresponds to the dimensionality of each layer, and both horizontal and vertical dimensions are convolutional. This network takes input $x \in \mathbb{R}^{T \times U \times 1}$, and applies the following operations: $d_1 = 32$ convolutional filters (13 frames by 13 frequency bins) with ReLU activations, $d_2 = 24$ convolutional filters (9×9), 3×3 max pooling, $d_4 = 24$ convolutional filters (1×5), and a softmax output layer (all 73 vertical positions) over 16 classes

When the desired output of the model is a time-varying prediction, it is common to introduce a full-height layer (i.e., $p_k = U_{k-1}$), so that all subsequent layers effectively become one-dimensional convolutions over the time dimension. Just as with the one-dimensional case (Sect. 5.4.3.1), global pooling or hybrid architectures (Sect. 5.4.5) can be used in settings that call for fixed-dimensional outputs.

5.4.4 *Recurrent Networks*

The convolutional networks described in the previous section are effective at modeling short-range interactions, due to their limited spatial locality. While pooling operators can expand the receptive field of convolutional filters, they are still not well-suited to modeling long-range interactions, or interactions with variable-length dependencies, which are common in certain forms of audio (e.g., spoken language or music). Recurrent networks [43, 100, 120] provide a more flexible framework in which to model sequential data. Rather than modeling a finite receptive field, observations are encoded and accumulated over temporal or spatial dimensions as latent state variables.

5.4.4.1 Recursive Networks

Like MLPs and convolutional networks, recurrent networks—also called recurrent neural networks (RNNs)—are built up by layers of processing units. In the simplest generic form, a recurrent layer defines a *state vector* $h[t] \in \mathbb{R}^{d_i}$ (at time index t), which is driven by input $z[t] \in \mathbb{R}^{d_{i-1}}$ and the state vector at the previous time $h[t-1]$:

$$h[t] := \rho\left(w^{\mathsf{T}} z[t] + v^{\mathsf{T}} h[t-1] + b\right), \tag{5.58}$$

where the layer parameters $\theta = (w, v, b)$ consist of *input weights* $w \in \mathbb{R}^{d_{i-1} \times d_i}$, *recurrent weights* $v \in \mathbb{R}^{d_i \times d_i}$, bias vector $b \in \mathbb{R}^{d_i}$, and element-wise non-linearity ρ. The model is *recurrent* because the state at time t depends on the state at time $t-1$ (which in turn depends on $t-2$, and so on) through the recurrent weights v, which play a role similar to the transition matrix in a hidden Markov model (5.37).[16] A recurrent layer therefore integrates information over all $t' \le t$ to produce the output state vector h_t at time t, and can thus be used to model sequential data with variable-length dependencies. The initial hidden state $h[0]$ is typically set to the all-zeros vector, so that $h[1] = \rho\left(w^{\mathsf{T}} z[1] + b\right)$ is driven only by the input and bias.

Just as with MLPs or convolutional networks, recursive networks can be stacked in a hierarchy of layers. The output of a recurrent layer $f_i : \mathbb{R}^{T \times d_{i-1}} \to \mathbb{R}^{T \times d_i}$ is the sequence of state vectors:

$$f_i(z \mid \theta) := (h[t])_{t=1}^{T}, \tag{5.59}$$

which can in turn be used as inputs to a second recursive layer, or to a convolutional layer which maps the hidden state vectors $h[t]$ to predicted outputs.

Learning the weights w and v for a recurrent layer is computationally challenging, since the gradient calculation depends on the entire state sequence. The standard practical approach to this problem is *back-propagation through time* (BPTT) [121], which approximates the full gradient by unrolling (5.58) up to a finite number k of time steps, and applying standard back-propagation to estimate gradients over length-k sub-sequences of the input. The BPTT approach for standard recurrent networks is known to suffer from the *vanishing and exploding gradient problem*, due to the cumulative effect of iteratively applying the state transformation v [10, 90]. In practice, this can limit the applicability of the recurrent formulation defined in (5.58) to relatively short sequences, though attempts have been made to apply the method to sequential tasks such as musical chord recognition [19] or phoneme sequence modeling [20]. For a comprehensive introduction to recursive networks and their challenges, we refer readers to Graves [56].

5.4.4.2 Gated Recurrent Units

The recently proposed gated recurrent unit (GRU) [25] architecture was explicitly designed to mitigate the challenges of gradient-based training of recurrent networks described in the previous section. Although the GRU architecture was proposed as a simplification of the long short-term memory (LSTM) architecture (Sect. 5.4.4.3), we present it here first to ease exposition.

Formally, a GRU consists of a *reset vector* $r[t] \in \mathbb{R}^{d_i}$ and an *update vector* $u[t] \in \mathbb{R}^{d_i}$, in addition to the hidden state vector $h[t] \in \mathbb{R}^{d_i}$. The state equations

[16]A key distinction between recurrent networks and HMMs is that the "state space" in a recurrent network is continuous, i.e., $h_t \in \mathbb{R}^{d_i}$.

are defined as follows:

$$r[t] := \sigma\left({w_r}^{\mathsf{T}} z[t] + {v_r}^{\mathsf{T}} h[t-1] + b_r\right) \tag{5.60a}$$

$$u[t] := \sigma\left({w_u}^{\mathsf{T}} z[t] + {v_u}^{\mathsf{T}} h[t-1] + b_u\right) \tag{5.60b}$$

$$\hat{h}[t] := \rho\left({w_h}^{\mathsf{T}} z[t] + {v_h}^{\mathsf{T}} \left(r[t] \odot h[t-1]\right) + b_h\right) \tag{5.60c}$$

$$h[t] := u[t] \odot h[t-1] + (1 - u[t]) \odot \hat{h}[t], \tag{5.60d}$$

where $\odot$ denotes the element-wise (Hadamard) product, σ is the logistic function, and the weights $w_r, w_u, w_h \in \mathbb{R}^{d_{i-1} \times d_i}$ and $v_u, v_r, v_h \in \mathbb{R}^{d_i \times d_i}$ and biases $b_r, b_u, b_h \in \mathbb{R}^{d_i}$ are all defined analogously to the standard recurrent layer (5.58).The transfer function ρ in (5.60c) is typically taken to be tanh. Non-saturating transfer functions are discouraged for this setting because they allow $h[t]$, and thus $v^{\mathsf{T}} h[t-1]$ to grow without bound, which in turn causes both exploding gradients (on v weights) and can limit the influence of the inputs $z[t]$ in the update equations.

The *gate* variables $r[t]$ and $u[t]$ control the updates to the state vector $h[t]$, which is a convex combination of the previous state $h[t-1]$ and a proposed next state $\hat{h}[t]$. When the update vector $u[t]$ is close to 1, (5.60d) persists the previous state $h[t-1]$ and discards the proposed state $\hat{h}[t]$. When $u[t]$ is close to 0 and $r[t]$ is close to 1, (5.60d) updates $h[t]$ according to the standard recurrent layer equation (5.58). When both $u[t]$ and $r[t]$ are close to 0, $h[t]$ "resets" to $\rho(w_h^{\mathsf{T}} z[t] + b_h)$, as if $z[t]$ was the first observation in the sequence. As in (5.59), the output of a GRU layer is the concatenation of hidden state vectors $(h[t])_{t=1}^{T}$.

Like a standard recurrent network, GRUs are also trained using the BPTT method. However, because a GRU can persist state vectors across long extents—when u_t stays near 1—the hidden state h_t does not result directly from successive applications of the transformation matrix v, so it is less susceptible to the vanishing/exploding gradient problem. Similarly, the reset variables allow the GRU to discard state information once it is no longer needed. Consequently, GRUs have been demonstrated to perform well for long-range sequence modeling tasks, such as machine translation [26].

5.4.4.3 Long Short-Term Memory Networks

Long short-term memory (LSTM) networks [66] were proposed long before the gated recurrent unit model of the previous section, but are substantially more complicated. Nonetheless, the LSTM architecture has been demonstrated to be effective for modeling long-range sequential data [112].

An LSTM layer consists of three *gate* vectors: the *input gate* $i[t]$, the *forget gate* $f[t]$, and the *output gate* $o[t]$, as well as the *memory cell* $c[t]$, and the state vector $h[t]$.

Following the formulation of Graves [57], the updates are defined as follows[17]:

$$i[t] := \sigma\left(w_i^\mathsf{T} z[t] + v_i^\mathsf{T} h[t-1] + b_i\right) \tag{5.61a}$$

$$f[t] := \sigma\left(w_f^\mathsf{T} z[t] + v_f^\mathsf{T} h[t-1] + b_f\right) \tag{5.61b}$$

$$o[t] := \sigma\left(w_o^\mathsf{T} z[t] + v_o^\mathsf{T} h[t-1] + b_o\right) \tag{5.61c}$$

$$\hat{c}[t] := \rho_c\left(w_c^\mathsf{T} z[t] + v_c^\mathsf{T} h[t-1] + b_c\right) \tag{5.61d}$$

$$c[t] := f[t] \odot c[t-1] + i[t] \odot \hat{c}[t] \tag{5.61e}$$

$$h[t] := o[t] \odot \rho_h\left(c[t]\right). \tag{5.61f}$$

The memory cell and all gate units have the standard recurrent net parameters $w \in \mathbb{R}^{d_{i-1} \times d_i}$, $v \in \mathbb{R}^{d_i \times d_i}$, $b \in \mathbb{R}^{d_i}$.

Working backward through the equations, the hidden state $h[t]$ (5.61f) can be interpreted as a point-wise non-linear transformation of the memory cell $c[t]$, which has been masked by the output gate $o[t]$. The output gate (5.61c) thus limits which elements of the memory cell are propagated through the recurrent updates (5.61a–5.61c). This is analogous to the reset functionality of a GRU.

The memory cell $c[t]$ (5.61e) behaves similarly to the hidden state $h[t]$ of the GRU (5.60d), except that "update" variable has been decoupled into the *input* and *forget* gates $i[t]$ and $f[t]$. When the forget gate $f[t]$ is low, it masks out elements from the previous memory cell value $c[t-1]$; when the input gate $i[t]$ is high, it integrates the proposed value $\hat{c}[t]$. Because $f[t]$ and $i[t]$ do not necessarily sum to 1, an additional transfer function ρ_h is included in (5.61f) to preserve boundedness of the hidden state vector $h[t]$. As in the GRU, tanh is the typical choice for the transfer functions ρ_h and ρ_c.

Recently, two empirical studies have studied the importance of the various components of the LSTM architecture [60, 72]. Taken together, their findings indicate that the forget gate $f[t]$ is critical to modeling long-range interactions. In particular, Józefowicz et al. note that it is helpful to initialize the bias term b_f to be relatively large (at least 1) so that the $f[t]$ stays high (propagates previous state) early in training [72]. Both studies also found that across several tasks, the simplified "update" logic of the GRU performs comparably to the more elaborate forget/input logic of the standard LSTM.

[17]The presentation of Graves [57] differs slightly in its inclusion of "peephole" connections [51]. We omit these connections here for clarity of presentation, and because recent studies have not demonstrated their efficacy [60].

5.4.4.4 Bi-directional Networks

The RNN, GRU, and LSTM architectures described in the previous sections are all designed to integrate information in one direction along a particular dimension, e.g., forward in time. In some applications, it can be beneficial to integrate information across both directions. Bi-directional recurrent networks (BRNNs) achieve this by a simple reduction to the standard one-directional architecture [104].

A BRNN layer $f_i(z \mid \theta)$ consists of two standard recurrent layers: the *forward layer* $\overrightarrow{f_i}$ and the *backward layer* $\overleftarrow{f_i}$. The BRNN layer $f_i : \mathbb{R}^{T \times d_{i-1}} \to \mathbb{R}^{T \times d_i}$ is the concatenation:

$$f_i(z \mid \theta) := \begin{bmatrix} \overrightarrow{f_i}(z \mid \theta) \\ \operatorname{rev}\left(\overleftarrow{f_i}(\operatorname{rev}(z) \mid \theta)\right) \end{bmatrix}, \tag{5.62}$$

where $\operatorname{rev}(\cdot)$ reverses its input along the recurrent dimension, and d_i combines the dimensionality of the forward and backward layers.[18] The output $f_i(z \mid \theta)[t]$ at time t thus includes information integrated across the entire sequence, before and after t. This approach can be easily adapted to LSTM [59] and GRU architectures [6].

Bi-directional networks—in particular, the B-LSTM approach—have been demonstrated to be effective for a wide range of audio analysis tasks, including beat tracking [17], speech recognition [58, 84], and event detection [89]. Unless the application requires forward-sequential processing, e.g., online prediction, bi-directional networks are strictly more powerful, and generally preferred.

5.4.5 *Hybrid Architectures*

The previous sections covered a range of architectural units for deep networks, but in practice, these architectures are not often used in isolation. Instead, practitioners often design models using combinations of the techniques described above. Here, we briefly summarize the two most commonly used hybrid architectures.

5.4.5.1 Convolutional + Dense

As briefly mentioned in Sect. 5.4.3, many applications consist of variable-length input with fixed-length output, e.g., assigning a single label to an entire audio excerpt. This presents a problem for purely convolutional architectures, where in the absence of global pooling operators, the output length is proportional to the input

[18]Some authors define the BRNN output (5.62) as a non-linear transformation of the concatenated state vectors [55]. This formulation is equivalent to (5.62) followed by a one-dimensional convolutional layer with a receptive field $n_i = 1$, so we opt for the simpler definition here.

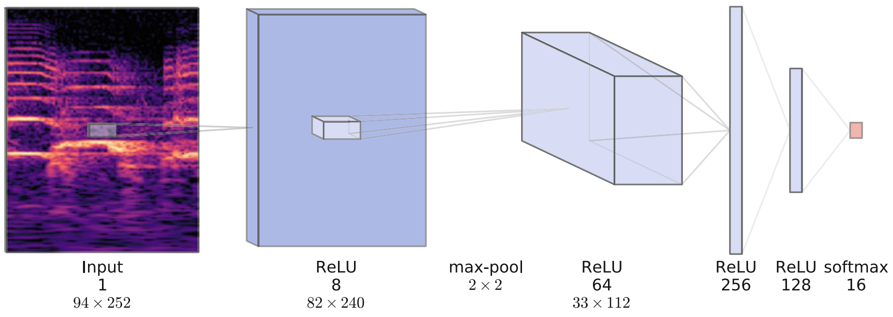

Fig. 5.8 An example of a convolutional-dense architecture. Two convolutional layers are followed by two dense layers and a 16-class output layer

length (or spatial extents, in two-dimensional convolutional models). While global pooling operators can resolve this by aggregating over the entirety of the variable-length input dimensions, the resulting summaries cannot model the dynamics of interactions between features over the convolutional dimension. As a concrete example,

$$\max([0,1]) = \max([1,0]) = 1 \tag{5.63}$$

discards the ordering information of the input, which may be relevant for describing certain phenomena.

A common solution to this problem is to replace global pooling operators with dense connections—i.e., one or more MLP layers, also called "fully connected" in this context—to map convolutional outputs to a fixed-dimensional representation. An example of this type of architecture is illustrated in Fig. 5.8.

Note that the convolutional-dense architecture requires all input data x be truncated or padded to a fixed length. Consequently, when deploying the resulting model, the input data must be sliced into fixed-length observation windows, and the resulting window predictions must be collected and aggregated to produce the final prediction. With this approach, care must be taken to ensure that the model evaluation corresponds to quantity of interest (e.g., recording-level rather than window-level prediction). Nonetheless, the general convolutional-dense approach has been demonstrated to perform well on a wide array of tasks [37, 93, 102, 116].

5.4.5.2 Convolutional + Recurrent

Another increasingly common hybrid architecture is to follow one or more convolutional layers by recurrent layers. This approach—alternately known as *convolutional encoder-recurrent decoder*, or *convolutional-recurrent neural network* (CRNN)—combines local feature extraction with global feature integration. Although this architecture is only recently becoming popular, it has been demonstrated to be effective in several applications, including speech recognition [3], image recognition [123], text analysis [113], and music transcription [108].

5.5 Improving Model Stability

In audio applications, the efficacy of a model is often impeded by limited access to a large, well-annotated, and representative sample of training data. Models trained on small, biased, or insufficiently varied data sets can over-fit and generalize poorly to unseen data. Practitioners have developed several techniques to mitigate this issue, and in this section, we briefly summarize three important concepts: data augmentation, domain adaptation, and ensemble methods.

5.5.1 Data Augmentation

Data augmentation is an increasingly popular method to help mitigate sampling bias: the general idea is to perturb the training data during training to inject more variance into the sample. By doing so, the model is exposed to a larger and more varied training sample, and may therefore be better able to characterize the decision boundaries between classes.

Perturbations can range from simple effects like additive background noise, constant pitch shifting, and time stretching [83, 102], to more sophisticated, domain-specific deformations like vocal tract length perturbation [34, 70] or variable speed perturbation [76]. In general, the idea is that training on these modified examples can help the model become invariant to the chosen deformations. Care must be taken to ensure that the deformations applied to the input audio leave the labels unmodified (or at least modified in a controlled way) [83].

5.5.2 Domain Adaptation

Throughout this chapter, there has been an underlying assumption that the training sample S is drawn I.I.D. from the test distribution $\mathscr{D}$. In practice, this assumption is often violated. Training data can be biased, e.g., when labeled examples produced in one recording environment are used to develop a model which is deployed in a different environment. In general, this problem is known as *domain adaptation* [16]: a model is trained on a sample S drawn from a different domain (distribution) than the eventual target domain.

In general, methods for domain adaptation require access to a labeled training set S drawn from a source distribution $\mathscr{D}_s$, and an unlabeled sample S' drawn from the target distribution $\mathscr{D}$. The majority of domain adaptation techniques operate by example weighting. These methods replace the unweighted sum in the learning objective (5.6) by a weighted sum so that it better approximates the loss on the target distribution $\mathscr{D}$ [14, 29, 61]. Alternatively, feature transformation methods distort the training data features so that it is difficult to distinguish samples of $\mathscr{D}_s$ from those of $\mathscr{D}$ [46, 49, 53]. In both cases, the underlying goal is to minimize the discrepancy between the training distribution (or the loss estimated on the training sample) and the target distribution.

5.5.3 Ensemble Methods

Many of the methods described throughout this chapter can be sensitive to certain modeling decisions, such as input representation, network architecture, initialization schemes, or sampling of training data. As mentioned in the beginning of this chapter, a common remedy is to optimize over these decisions by using withheld validation set. However, this can still bias the resulting model if the validation sets are too small or insufficiently varied.

A complementary approach to combine multiple predictors $(h_1, \ldots, h_n)$ in an *ensemble*. There are many ways to go about this, such as majority voting over predictions, or weighted averaging over scores/likelihoods [21, 39]. In practice, when a single model appears to be reaching a performance plateau for a given task, an ensemble can often provide a modest improvement.

5.6 Conclusions and Further Reading

This chapter aimed to provide a high-level overview of supervised machine learning techniques for classification problems. When faced with a specific classification problem, the abundance and diversity of available techniques can present a difficult choice for practitioners. While there is no general recipe for selecting an appropriate method, there are several factors to consider in the process.

The first, and most important factor, is the availability and form of training data. Discriminative models generally require strongly labeled examples, both positively and negatively labeled. For example, in implementing a discriminative bird song detector, it's just as important to provide examples that do not include birds, so that the model can learn to discriminate. If negative examples are not available, a generative, class-conditional model may be more appropriate, but it may require a larger training sample than a discriminative method, due to the increased complexity of modeling the joint density $\mathbf{P}_\theta[x, y]$.

In audio applications, the characteristics of the target concept can also play an important role. Some concepts are obviously localized in time (e.g., transient sound events like gunshots), while others are diffused over long extents (e.g., in scene classification), and others are distinguished by dynamics over intermediate durations (e.g., musical rhythms). These characteristics should be taken into consideration when deciding between localized models (e.g., convolutional networks), dynamic models (HMMs or recurrent networks), or global models that integrate across the entirety of an observation (e.g., the bag-of-frames models described in Chap. 4).

Although machine learning algorithms are generally characterized by the loss functions they attempt to minimize, the treatment presented in this chapter only covers the relatively well-understood binary and multi-class categorization loss functions. In practical applications, the utility of a model can be measured according to a much broader space of evaluation criteria, which are discussed in Chap. 6. Bridging the gap between evaluation and modeling is an area of active research, which broadly falls under the umbrella of *structured output prediction* in the machine learning literature [78, 115].

Additionally, this chapter presents the simplest learning paradigm, in which a fully annotated sample is available for parameter estimation. In reality, a variety of learning paradigms exist, each making different assumptions about the training and test data. These formulations include: *semi-supervised learning* [24], where unlabeled observations are also available; *multiple-instance learning*, where a positive label is applied to a collection of observations, indicating at least one of which is a positive example [40]; and *positive-unlabeled* learning, where labels are only available for a subset of the target concepts, and unlabeled examples may or may not belong to those classes [42]. The choice of learning paradigm ultimately derives from the form of training data available, and how the resulting model will be deployed to make predictions.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). http://tensorflow.org/, Software available from tensorflow.org
2. Akaike, H.: Likelihood of a model and information criteria. J. Econom. **16**(1), 3–14 (1981)
3. Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., et al.: Deep speech 2: end-to-end speech recognition in English and Mandarin (2015). arXiv preprint arXiv:1512.02595
4. Andrieu, C., De Freitas, N., Doucet, A., Jordan, M.I.: An introduction to MCMC for machine learning. Mach. Learn. **50**(1–2), 5–43 (2003)
5. Antoniak, C.E.: Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. Ann. Stat. **2**, 1152–1174 (1974)
6. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate (2014). arXiv preprint arXiv:1409.0473
7. Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state Markov chains. Ann. Math. Stat. **37**(6), 1554–1563 (1966)
8. Beal, M.J.: Variational algorithms for approximate Bayesian inference. University of London (2003)
9. Bellet, A., Habrard, A., Sebban, M.: A survey on metric learning for feature vectors and structured data (2013). arXiv preprint arXiv:1306.6709
10. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)

11. Bergstra, J., Bastien, F., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., Desjardins, G., Warde-Farley, D., Goodfellow, I., Bergeron, A., et al.: Theano: deep learning on GPUs with python. In: Big Learn, Neural Information Processing Systems Workshop (2011)
12. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**(Feb), 281–305 (2012)
13. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)
14. Bickel, S., Brückner, M., Scheffer, T.: Discriminative learning under covariate shift. J. Mach. Learn. Res. **10**(Sep), 2137–2155 (2009)
15. Blei, D.M., Jordan, M.I., et al.: Variational inference for Dirichlet process mixtures. Bayesian Anal. **1**(1), 121–144 (2006)
16. Blitzer, J., McDonald, R., Pereira, F.: Domain adaptation with structural correspondence learning. In: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, pp. 120–128. Association for Computational Linguistics, Trento (2006)
17. Böck, S., Schedl, M.: Enhanced beat tracking with context-aware neural networks. In: Proceedings of the International Conference on Digital Audio Effects (2011)
18. Bottou, L.: Stochastic gradient learning in neural networks. Proc. Neuro-Nımes **91**(8), 687–696 (1991)
19. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Audio chord recognition with recurrent neural networks. In: Proceedings of the International Conference on Music Information Retrieval, pp. 335–340. Citeseer (2013)
20. Boulanger-Lewandowski, N., Droppo, J., Seltzer, M., Yu, D.: Phone sequence modeling with recurrent neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5417–5421. IEEE, New York (2014)
21. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
22. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and regression trees. CRC Press, New York (1984)
23. Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M.A., Guo, J., Li, P., Riddell, A.: Stan: a probabilistic programming language. J. Stat. Softw. **20**, 1–37 (2016)
24. Chapelle, O., Scholkopf, B., Zien, A.: Semi-supervised learning. IEEE Trans. Neural Netw. **20**(3), 542–542 (2009)
25. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, pp. 1724–1734 (2014)
26. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: encoder-decoder approaches (2014). arXiv preprint arXiv:1409.1259
27. Chollet, F.: Keras. https://github.com/fchollet/keras (2015). Retrieved on 2017-01-02.
28. Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: a matlab-like environment for machine learning. In: Big Learn, Neural Information Processing Systems Workshop, EPFL-CONF-192376 (2011)
29. Cortes, C., Mohri, M.: Domain adaptation and sample bias correction theory and algorithm for regression. Theor. Comput. Sci. **519**, 103–126 (2014)
30. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
31. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley, New York (2012)
32. Cox, D.R.: The regression analysis of binary sequences. J. R. Stat. Soc. Ser. B Methodol. **20**, 215–242 (1958)
33. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. J. Mach. Learn. Res. **2**(Dec), 265–292 (2001)
34. Cui, X., Goel, V., Kingsbury, B.: Data augmentation for deep neural network acoustic modeling. IEEE/ACM Trans. Audio Speech Lang. Process. (TASLP) **23**(9), 1469–1477 (2015)

35. Dasarathy, B.V.: Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. IEEE Computer Society Press, Los Alamitos (1991)
36. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. Ser. B (Methodological) **39**, 1–38 (1977)
37. Dieleman, S., Schrauwen, B.: End-to-end learning for music audio. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6964–6968. IEEE, New York (2014)
38. Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S.K., Nouri, D., Maturana, D., Thoma, M., Battenberg, E., Kelly, J., Fauw, J.D., Heilman, M., Diogo149, McFee, B., Weideman, H., Takacsg84, Peterderivaz, Jon, Instagibbs, Rasul, D.K., CongLiu, Britefury, Degrave, J.: Lasagne: first release (2015). doi:10.5281/zenodo.27878. https://doi.org/10.5281/zenodo.27878
39. Dietterich, T.G.: Ensemble learning. In: The Handbook of Brain Theory and Neural Networks, 2nd edn., pp. 110–125. MIT Press, Cambridge, MA (2002)
40. Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artif. Intell. **89**(1), 31–71 (1997)
41. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(Jul), 2121–2159 (2011)
42. Elkan, C., Noto, K.: Learning classifiers from only positive and unlabeled data. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 213–220. ACM, New York (2008)
43. Elman, J.L.: Finding structure in time. Cogn. Sci. **14**(2), 179–211 (1990)
44. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: a library for large linear classification. J. Mach. Learn. Res. **9**(Aug), 1871–1874 (2008)
45. Feldman, V., Guruswami, V., Raghavendra, P., Wu, Y.: Agnostic learning of monomials by halfspaces is hard. In: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science, pp. 385–394. IEEE Computer Society, New York (2009)
46. Fernando, B., Habrard, A., Sebban, M., Tuytelaars, T.: Unsupervised visual domain adaptation using subspace alignment. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2960–2967 (2013)
47. Fix, E., Hodges, J.L. Jr.: Discriminatory analysis-nonparametric discrimination: consistency properties. Technical Report, DTIC Document (1951)
48. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning. Springer Series in Statistics, vol. 1. Springer, Berlin (2001)
49. Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., Lempitsky, V.: Domain-adversarial training of neural networks. J. Mach. Learn. Res. **17**(59), 1–35 (2016). http://jmlr.org/papers/v17/15-239.html
50. Gelfand, A.E., Smith, A.F.: Sampling-based approaches to calculating marginal densities. J. Am. Stat. Assoc. **85**(410), 398–409 (1990)
51. Gers, F.A., Schmidhuber, J.: Recurrent nets that time and count. In: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, vol. 3, pp. 189–194. IEEE, New York (2000)
52. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: International Conference on Artificial Intelligence and Statistics, vol. 9, pp. 249–256 (2010)
53. Gong, B., Shi, Y., Sha, F., Grauman, K.: Geodesic flow kernel for unsupervised domain adaptation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2066–2073 (2012)
54. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA (2016). http://www.deeplearningbook.org
55. Graves, A.: Sequence transduction with recurrent neural networks. CoRR abs/1211.3711 (2012). http://arxiv.org/abs/1211.3711
56. Graves, A.: Supervised Sequence Labelling with Recurrent Neural Networks. Springer, Berlin (2012)

57. Graves, A.: Generating sequences with recurrent neural networks (2013). arXiv preprint arXiv:1308.0850
58. Graves, A., Jaitly, N.: Towards end-to-end speech recognition with recurrent neural networks. In: Proceedings of the International Conference on Machine Learning, vol. 14, pp. 1764–1772 (2014)
59. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. **18**(5), 602–610 (2005)
60. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: a search space odyssey (2015). arXiv preprint arXiv:1503.04069
61. Gretton, A., Smola, A., Huang, J., Schmittfull, M., Borgwardt, K., Schölkopf, B.: Covariate shift by kernel mean matching. Dataset Shift Mach. Learn. **3**(4), 5 (2009)
62. Hastings, W.K.: Monte carlo sampling methods using Markov chains and their applications. Biometrika **57**(1), 97–109 (1970)
63. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034 (2015)
64. Heittola, T., Mesaros, A., Eronen, A., Virtanen, T.: Context-dependent sound event detection. EURASIP J. Audio Speech Music Process. **2013**(1), 1–13 (2013)
65. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
66. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
67. Humphrey, E.J., Bello, J.P.: Rethinking automatic chord recognition with convolutional neural networks. In: 2012 11th International Conference on Machine Learning and Applications (ICMLA), vol. 2, pp. 357–362. IEEE, New York (2012)
68. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pp. 448–456 (2015)
69. Ishwaran, H., Zarepour, M.: Exact and approximate sum representations for the Dirichlet process. Can. J. Stat. **30**(2), 269–283 (2002)
70. Jaitly, N., Hinton, G.E.: Vocal tract length perturbation (VTLP) improves speech recognition. In: Proceedings of ICML Workshop on Deep Learning for Audio, Speech and Language (2013)
71. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: convolutional architecture for fast feature embedding (2014). arXiv preprint arXiv:1408.5093
72. Józefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, 6–11 July 2015, pp. 2342–2350 (2015). http://jmlr.org/proceedings/papers/v37/jozefowicz15.html
73. Jurafsky, D., Martin, J.H.: Speech and language processing: an introduction to speech recognition. Computational Linguistics and Natural Language Processing. Prentice Hall, Upper Saddle River (2008)
74. Kearns, M.J.: The Computational Complexity of Machine Learning. MIT Press, Cambridge (1990)
75. Kingma, D., Ba, J.: Adam: a method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980
76. Ko, T., Peddinti, V., Povey, D., Khudanpur, S.: Audio augmentation for speech recognition. In: Proceedings of INTERSPEECH (2015)
77. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**(1), 79–86 (1951)

78. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of the International Conference on Machine Learning, ICML, vol. 1, pp. 282–289 (2001)
79. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
80. Lee, H., Pham, P., Largman, Y., Ng, A.Y.: Unsupervised feature learning for audio classification using convolutional deep belief networks. In: Advances in Neural Information Processing Systems, pp. 1096–1104 (2009)
81. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of ICML Workshop on Deep Learning for Audio, Speech, and Language Processing (2013)
82. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, vol. 1, no. 14, pp. 281–297 (1967)
83. McFee, B., Humphrey, E.J., Bello, J.P.: A software framework for musical data augmentation. In: International Society for Music Information Retrieval Conference (ISMIR) (2015)
84. Mesnil, G., He, X., Deng, L., Bengio, Y.: Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In: Proceedings of INTERSPEECH, pp. 3771–3775 (2013)
85. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge (2012)
86. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807–814 (2010)
87. Neal, R.M.: Probabilistic inference using Markov chain monte carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, Toronto, Ontario (1993)
88. Nesterov, Y.: A method of solving a convex programming problem with convergence rate O (1/k2). Sov. Math. Dokl. **27**(2), 372–376 (1983)
89. Parascandolo, G., Huttunen, H., Virtanen, T.: Recurrent neural networks for polyphonic sound event detection in real life recordings. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6440–6444. IEEE, New York (2016)
90. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Proceedings of the 30th International Conference on Machine Learning, ICML (3), vol. 28, pp. 1310–1318 (2013)
91. Pearson, K.: Contributions to the mathematical theory of evolution. Philos. Trans. R. Soc. Lond. A **185**, 71–110 (1894)
92. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**(Oct), 2825–2830 (2011)
93. Piczak, K.J.: Environmental sound classification with convolutional neural networks. In: 2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP), pp. 1–6. IEEE, New York (2015)
94. Platt, J., et al.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Adv. Large Margin Classif. **10**(3), 61–74 (1999)
95. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Comput. Math. Math. Phys. **4**(5), 1–17 (1964)
96. Rabiner, L., Juang, B.: An introduction to hidden Markov models. IEEE ASSP Mag. **3**(1), 4–16 (1986)
97. Raiffa, H.: Bayesian decision theory. Recent Developments in Information and Decision Processes, pp. 92–101. Macmillan, New York (1962)
98. Rasmussen, C.E.: The infinite Gaussian mixture model. In: Neural Information Processing Systems, vol. 12, pp. 554–560 (1999)
99. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386 (1958)

100. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cogn. Model. **5**(3), 1 (1988)
101. Rumelhart, D.E., McClelland, J.L., Group, P.R., et al.: Parallel Distributed Processing, vol. 1. IEEE, New York (1988)
102. Schlüter, J., Grill, T.: Exploring data augmentation for improved singing voice detection with neural networks. In: 16th International Society for Music Information Retrieval Conference (ISMIR-2015) (2015)
103. Schölkopf, B., Herbrich, R., Smola, A.J.: A generalized representer theorem. In: International Conference on Computational Learning Theory, pp. 416–426. Springer, London (2001)
104. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Trans. Signal Process. **45**(11), 2673–2681 (1997)
105. Schwarz, G., et al.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)
106. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, Cambridge (2014)
107. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
108. Sigtia, S., Benetos, E., Dixon, S.: An end-to-end neural network for polyphonic piano music transcription. IEEE/ACM Trans. Audio Speech Lang. Process. **24**(5), 927–939 (2016)
109. Sjöberg, J., Ljung, L.: Overtraining, regularization and searching for a minimum, with application to neural networks. Int. J. Control. **62**(6), 1391–1407 (1995)
110. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
111. Sutskever, I., Martens, J., Dahl, G.E., Hinton, G.E.: On the importance of initialization and momentum in deep learning. In: Proceedings of the International Conference on International Conference on Machine Learning, ICML (3), vol. 28, pp. 1139–1147 (2013)
112. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)
113. Tang, D., Qin, B., Liu, T.: Document modeling with gated recurrent neural network for sentiment classification. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1422–1432 (2015)
114. Tran, D., Kucukelbir, A., Dieng, A.B., Rudolph, M., Liang, D., Blei, D.M.: Edward: a library for probabilistic modeling, inference, and criticism (2016). arXiv preprint arXiv:1610.09787
115. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. J. Mach. Learn. Res. **6**(Sep), 1453–1484 (2005)
116. Van den Oord, A., Dieleman, S., Schrauwen, B.: Deep content-based music recommendation. In: Advances in Neural Information Processing Systems, pp. 2643–2651 (2013)
117. Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inf. Theory **13**(2), 260–269 (1967)
118. Wainwright, M.J., Jordan, M.I.: Graphical models, exponential families, and variational inference. Found. Trends Mach. Learn. **1**(1–2), 1–305 (2008)
119. Watanabe, S.: Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. J. Mach. Learn. Res. **11**(Dec), 3571–3594 (2010)
120. Werbos, P.J.: Generalization of backpropagation with application to a recurrent gas market model. Neural Netw. **1**(4), 339–356 (1988)
121. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proc. IEEE **78**(10), 1550–1560 (1990)
122. Zadrozny, B., Elkan, C.: Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 694–699. ACM, New York (2002)
123. Zuo, Z., Shuai, B., Wang, G., Liu, X., Wang, X., Wang, B., Chen, Y.: Convolutional recurrent neural networks: learning spatial dependencies for image representation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 18–26 (2015)