An introduction to structured discriminative learning

Roland Memisevic
Department of Computer Science,
University of Toronto,
roland@cs.toronto.edu

December 21, 2006

Abstract

We provide a tutorial overview of supervised learning with structured outputs. Taking the perspective of linear classification, we describe several recently discussed approaches in a unified framework, in which particular instantiations differ only in the choice of loss-function. We describe in detail the problems of parameter estimation and inference in these models and discuss nonparametric variants that are based on the use of kernels.

1 Introduction

Machine learning offers two complementary strategies for constructing intelligent systems: The first is to construct them 'by example' in order to overcome the need for tedious hand-coding; the second is to use a 'divide-and-conquer' approach, that helps to deal with uncertainty and brittleness when building complex systems from many smaller components.

The first strategy is usually associated with supervised, or discriminative, learning: Training data is used to specify a desired input-output behavior, and learning is used to adapt parameters of the model, so that it behaves 'similarly' on unseen inputs in the future. Of crucial importance for this strategy is the issue of *generalization*: Only if the system is likely to behave as desired on unseen test data, can this strategy be useful. Generalization, and conditions that warrant it, are therefore the central issue in the theory of supervised learning.

Discriminative methods are usually defined for problems such as classification and regression, that is problems where the output is a class-label or a typically low-dimensional real vector. Most real world tasks are more complex than these, and require systems that are composed of many subsystems. To understand a spoken utterance in a speech recognition task, for example, we need to perform segmentation on the signal level, recognize phonemes, morphemes and words, parse these constituents to obtain higher level structures, and combine these with semantic and possibly also context information. One way of solving such a complex task would be to construct a collection of models, one for each subtask, and to combine these to obtain the complete system. However, since subtasks are usually intertwined and strongly interdependent, combining and making sub-models work nicely together is an extremely difficult problem, for which the standard methods in supervised learning usually do not provide a solution.

This problem of combining many 'local' models into a single 'global' model, is a problem for which typically unsupervised methods provide a solution. A standard approach in this area is to define a joint probability distribution over a collection of random variables that represent quantities involved in the task. Formally, such a distribution can often be specified using a probabilistic graphical model. By being probabilistic, such a model overcomes the problem of brittleness that complicates the construction of complex systems, and since it defines a single *joint* model over all components, it allows each component to constrain each other, and thereby forces them to work together nicely as desired.

A standard way of setting up a probabilistic system is by constructing a parametric model of the statistical process that generated the input data. Formally, such a model is defined as a conditional distribution of the data, where the conditioning variables are the system variables. Training such a *generative model* then amounts to maximizing with respect to the model parameters the probability of generating the training data. Applying it amounts to performing *inference* in the model in order to obtain the conditional distribution over the system variables given the input, and usually requires invoking Bayes' rule. The rationale behind this approach is that if the model is a good model of the data generating process, then the system variables will provide a good internal representation of the input, and so they will be useful in solving the complex task for which the model was designed. When the number of variables in the model is large, the distributions can be difficult to compute or to represent, since the number of joint instantiations over all variables is exponential in the number of variables. The standard approach for dealing with this problem is to make *conditional independence* assumptions, that allow us to factorize a given distribution into functions that are defined over subsets of variables. Many computations in the model can then – despite the exponential explosion – be performed efficiently using dynamic programming procedures. Alternatively, *variational methods* offer ways of approximating the intractable distribution by tractable ones.

While unsupervised learning methods have been successfully applied in a huge number of tasks, the unified treatment of input- and output-variables in a joint probabilistic framework has some disadvantages in practice. One problem is, that tractability considerations can force the system designer to make conditional independence assumptions that are undesirable. A related problem with generative models is, that they always *model* the input, even though an accurate model of the input is not necessary, if we are ultimately interested in modeling the *response* of the system to a given input. In that sense generative models are often not well 'tuned' to the problem at hand and can waste computational resources.

Recently, a general modeling framework has emerged, that combines ideas from both supervised and unsupervised learning. The methods are rooted in the 'engineering-by-example'-philosophy of supervised learning and can be described and analyzed in terms of generalization and regularization. At the same time they define complex, possibly probabilistic, models and allow us to apply dynamic programming and variational methods to perform inference. The basic idea underlying this framework is very simple: Rather than trying to *model* the input as would be done in a generative model, we consider systems that are *functions* of the input. In other words, we define the learning problem as the supervised problem of estimating a function that maps input data to a, possibly complex and probabilistic, model. The application of the model to test-data is then simply a problem in unsupervised inference.

Since the system response is modeled directly, many of the disadvantages of generative methods do not apply. Since the output, on the other hand, is a complex and structured model, and not a homogeneous object such as a single class label, more complex and structured problems can be addressed withing a single unified model than usually in supervised learning. In this review we will use the term 'structured response model' (SRM) to refer to these models.

The rest of this review is structured as follows: In the next section we describe in some detail standard approaches in supervised and unsupervised learning. One goal of this section, besides sketching the standard concepts of these areas, is to introduce the, not quite standard, notation, which has turned out to be useful for describing structured response models. We describe the models themselves in Section 3, where we also discuss the problems of parameter estimation and inference and further issues. In Section 4 we discuss open research problems and, as part of these, potential applications of structured response models. Section 5 is a short conclusion.

2 Learning and Inference in AI

2.1 Supervised Learning

In the following, we discuss supervised learning on the basis of (linear) classification and derive the two standard approaches from the perspective of *loss functions* for parameter estimation. We deviate from the standard notation used in the area in order to set the stage for the more complex models that we discuss in Section 3.

2.1.1 Linear classifiers

Formally, we can define the problem of supervised learning as follows: We are given a training set $\mathcal{D}$ of input-output pairs $\{(\mathbf{x}^i, y^i)\}_{i=1}^N$, where $\mathbf{x}^i$ denotes an input and y^i a corresponding output, and our goal is to infer a function $f(\mathbf{x})$ that behaves 'similarly' on test data as specified in the training set. Throughout we will concentrate on discrete output spaces for clarity and point to possible continuous extensions at the appropriate places in the text. In this section, we also restrict y to be scalar and assume that it takes on one out of K class-labels. Models that deal with vector-valued outputs are the subject of section 3.

The standard approach to solving the problem is to define a class $\mathcal{H}^{\mathbf{w}}$ of functions $f(\mathbf{x}; \mathbf{w})$ that are parameterized with some parameter vector $\mathbf{w}$. A particular instantiation can then be chosen from this class by optimizing the performance on the training data-set wrt. $\mathbf{w}$. A function class that will be of foremost interest throughout the rest of this review is the class of *linear classifiers*. Using a *joint*[1] *feature vector* $\phi(\mathbf{x}, y)$, we define the decision of a linear classifier as:

$$f(\mathbf{x}; \mathbf{w}) = \arg\max_y \mathbf{w}^T \phi(\mathbf{x}, y). \tag{1}$$

We can interpret the decision function as follows: The inner product $c(\mathbf{x}, y; \mathbf{w}) := \mathbf{w}^T \phi(\mathbf{x}, y)$ computes a *score*, that can be viewed as a measure of the compatibility between the input $\mathbf{x}$ and a potential class label y. The 'arg-max' then chooses that class which gives rise to the highest score. The class-dependent, joint feature representation $\phi(\mathbf{x}, y)$ can be used to capture properties on which the compatibility between input and output might depend. Its definition is part of the design of the classifier. A convenient property of this approach is that, since the input enters the classification decision only via the feature vector, we can define a classifier on arbitrary input spaces, beyond, say, real vector spaces. All we need to do is encode those properties of input-output pairs that we think might be useful for classification in the entries of the feature vector $\phi(\mathbf{x}, y)$.

One possible way of choosing an element from the function class $\mathcal{H}^{\mathbf{w}}$ is by measuring the performance on the training data. For this end we can consider the *empirical risk*, defined as

$$\mathcal{R}_{\mathcal{D}}(\mathbf{w}) = \frac{1}{N} \sum_i l(\mathbf{x}^i, y^i, \mathbf{c}(\mathbf{x}^i; \mathbf{w})), \tag{2}$$

which depends on a suitably chosen loss-function $l(\mathbf{x}^i, y^i, \mathbf{c}(\mathbf{x}^i, \mathbf{w}))$ that measures how undesirable the vector of scores $\mathbf{c}(\mathbf{x}^i, \mathbf{w}) := \big(c(\mathbf{x}^i, y; \mathbf{w})\big)_y$ is for training pair $(\mathbf{x}^i, y^i)$. A standard choice for l is the 0/1-loss which is zero for correctly classified examples (ie. examples for which $c_{y_i}(\mathbf{x}^i; \mathbf{w}) > c_y(\mathbf{x}^i; \mathbf{w}) \quad \forall y \neq y^i$) and one for the others:

$$l^{0/1}(\mathbf{x}^i, y^i, \mathbf{c}(\mathbf{x}^i; \mathbf{w})) = 1 - \delta_{y^i, f(\mathbf{x}^i; \mathbf{w})}, \tag{3}$$

where $\delta_{y,y'}$ is the Kronecker-delta. Under this loss, the empirical risk simply counts the relative number of wrongly classified examples. Many other loss-functions are possible, and we will discuss several alternatives later on.

While minimizing the empirical risk (Eq. 2) allows us to optimize the performance on the training data, in practice we are usually interested in good performance on potential *test* cases. That the empirical risk is *related* in some way to the performance on unseen test cases is obvious. Under what circumstances and to what degree a small empirical risk does indeed entail good performance on test data, is the subject of statistical learning theory [69]. Assuming the training cases have been drawn independently from some distribution $p(\mathbf{x}, y)$, the central results of this area concern how the *expected error* wrt. this distribution,

$$\mathcal{R}_{\infty}(\mathbf{w}) = \frac{1}{N} \sum_y \int p(\mathbf{x}, y) l(\mathbf{x}, y, \mathbf{c}(\mathbf{x}; \mathbf{w})) d\mathbf{x} \tag{4}$$

is related to the empirical risk. Bounds on the difference between the two quantities can be derived, that depend on quantities that measure the size of the function class under consideration or its complexity.

[1] In the literature, instead of using joint feature vectors $\phi(\mathbf{x}, y)$, usually only input features $\phi(\mathbf{x})$ are used. The multi-class decision function is then defined by using a set of weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_K$, ie. one for each class. We use the joint feature representation, because structured problems (Section 3) can be most conveniently described using these.

One such quantity is the VC-dimension. For some classifiers, such as the linear ones we consider here, a related quantity that is especially useful in practice is simply the squared 2-norm of the weight vector $\mathbf{w}$: The smaller $\|\mathbf{w}\|^2$, the better the generalization. Since good generalization to unseen data will be useful only if the system performs also well on the training data, the common strategy for training a classifier is to minimize the weighted sum:

$$\mathcal{R}_{\text{reg}}(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_i l(\mathbf{x}^i, y^i, \mathbf{c}(\mathbf{x}^i; \mathbf{w})), \tag{5}$$

where λ trades off generalization performance for classification performance on the training data. The practice of adding a penalty term such as $\|\mathbf{w}\|^2$ is referred to as *regularization* and the term itself as regularizer. A good value for λ is usually found by measuring performance on a hold-out set.

The optimization problem is defined up to the loss function $l(\mathbf{x}^i, y^i, \mathbf{c}(\mathbf{x}^i; \mathbf{w}))$, so to use this approach in practice, all we need to do is to plug in a suitable one. While the 0/1-loss is often a natural choice theoretically, it does not lend itself to efficient optimization, because of its discontinuity. To obtain an optimization problem that can be solved efficiently in practice, several continuous, or also differentiable, substitutes for the 0/1-loss are frequently used. We will discuss two important choices in the following two subsections. While they can both be viewed as ways of obtaining efficiently solvable versions of Problem 5, they also give rise to new perspectives on the learning problem itself. In particular, the first allows for an interpretation of the output of the classifier in probabilistic terms; the second one leads to the notion of margins, which provide a geometrical intuition onto the problem of regularization. A further difference in which both loss-functions differ from the 0/1-loss is that they not only assess the correctness of classification decisions, but also the *confidence* that the classifier has in these.

Another property of the 0/1-loss is that it treats all mis-classifications equally. Another reason to look for alternative loss functions could therefore be, that in a particular task some mis-classifications might be more problematic than others. A general way of dealing with this problem, is to define a *risk matrix* $\Delta(y, y')$ that associates a cost with each classification decision i where the correct class is j. Training a system that accommodates this definition of risk can be performed by extending the standard learning problem (Eq. 5) in suitable ways. We will discuss this issue in more detail in Section 3.

2.1.2 Probabilistic modeling and logistic regression

One way to obtain a differentiable alternative to the 0/1-loss is to reconsider the classification problem in a probabilistic setting ([31]), by defining the *conditional probability distribution* $p(y|\mathbf{x}; \mathbf{w})$. A canonical way of defining this distribution based on a linear compatibility measure (Eq. 1) is by setting:

$$p(y|\mathbf{x}; \mathbf{w}) = \frac{1}{Z(\mathbf{x}; \mathbf{w})} \exp(\mathbf{w}^T \phi(\mathbf{x}, y)), \quad \text{with} \tag{6}$$

$$Z(\mathbf{x}; \mathbf{w}) = \sum_y \exp(\mathbf{w}^T \phi(\mathbf{x}, y)), \tag{7}$$

where the exponential ensures positivity, and the (input-dependent) *partition function* $Z(\mathbf{x}; \mathbf{w})$ the normalization of the distribution. The model is commonly referred to as *logistic regression.* A typical inference problem under this model, based on a test input $\mathbf{x}^{\text{test}}$, is to find $f(\mathbf{x}^{\text{test}}) = \arg\max_y p(y|\mathbf{x}^{\text{test}}; \mathbf{w})$, which by taking the 'log' can be seen to be the problem in Eq. 1. However, having a distribution over y can be advantageous, since it allows us to compare confidences across potential candidate values, for example. Note also that we can view the minimizer of Eq. 5 under this loss as the maximum a posteriori (MAP)-estimate under a Bayesian model with Gaussian prior on $\mathbf{w}$.

A standard way of fitting the parameters of a probabilistic model is to maximize the conditional log-likelihood averaged over the training data:

$$\mathcal{L}(\mathbf{w}) = \sum_i \log p(y^i|\mathbf{x}^i; \mathbf{w}). \tag{8}$$

Note that maximizing a regularized version of this objective function is equivalent to minimizing Eq. 5, with a per-example 'log-'loss defined as

$$l^{\log}(\mathbf{x}^i, y^i, \mathbf{c}(\mathbf{x}^i; \mathbf{w})) = -\log p(y^i|\mathbf{x}^i; \mathbf{w}) = \log Z(\mathbf{x}^i, \mathbf{w}) - \mathbf{w}^T \phi(\mathbf{x}^i, y^i) \tag{9}$$

Since the log-loss is differentiable, we can optimize the model for example by using gradient-based methods. The objective function is convex, and so we are guaranteed to obtain the global optimum. The gradient of the log-likelihood can be easily shown to be

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \sum_i \phi(\mathbf{x}^i, y^i) - \mathrm{E}_{p(y|\mathbf{x}^i;\mathbf{w})}\phi(\mathbf{x}^i, y), \tag{10}$$

where $\mathrm{E}_{p(y|\mathbf{x}^i;\mathbf{w})}$ denotes expectation wrt. $p(y|\mathbf{x}^i;\mathbf{w})$. In standard classification problems, where the number of classes is relatively small, this expectation can be computed efficiently. The structured models that we discuss in Section 3.2 can be viewed as logistic regression models where the number of classes is huge. In that case, specialized procedures for computing this expectation, or at least approximating it, are required.

2.1.3 Direct optimization and support vector machines

Another continuous (but not differentiable) alternative to the 0/1-loss is the 'hinge-loss', which can be defined as

$$l^{\text{hinge}}(\mathbf{x}^i, y^i, c(\mathbf{x}^i, y^i; \mathbf{w})) = \max_y \Big(\mathbf{w}^T\phi(\mathbf{x}^i, y) + l^{0/1}(\mathbf{x}^i, y^i, \mathbf{c}(\mathbf{x}^i;\mathbf{w}))\Big) - \mathbf{w}^T\phi(\mathbf{x}^i, y^i). \tag{11}$$

The hinge loss measures the difference between the maximal confidence that the classifier has over all classes and the confidence it has in the correct class. In computing this maximum, all *wrong* classes get a 'head start' by adding 1 to the confidence. As a result, the hinge loss is 0, if the confidence in the correct class is by at least 1 greater than the confidence in the closest follow-up. Otherwise the loss scales linearly with the difference in these confidences.

Even though the hinge-loss is not differentiable, it can give rise to a tractable variant of the 0/1-loss based learning problem, too. The reason is that the hinge-loss allows us to recast the optimization problem (Eq. 5) as an equivalent *constrained optimization problem* ([17]; see [15],[18] for the binary setting): Introducing slack variables ξ_i, and the vector $\xi = (\xi_i)_i$, we can solve the problem implicitly by solving the quadratic program:

$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \tag{12}$$
$$\text{s.t.} \quad \mathbf{w}^T\phi(\mathbf{x}^i, y^i) - \max_y(\mathbf{w}^T\phi(\mathbf{x}^i, y) - \delta_{y,y^i}) = 1 - \xi_i, \quad \forall i,$$

where for consistency with the standard notation in the literature we use $C = \frac{1}{\lambda N}$ instead of λ (compare to Eq. 5) as a trade-off parameter here. To simplify the problem, note that we can replace each nonlinear equality constraint by N linear inequality constraints:

$$\mathbf{w}^T\phi(\mathbf{x}^i, y^i) - \mathbf{w}^T\phi(\mathbf{x}^i, y) \geq 1 - \delta_{y,y^i} - \xi_i, \quad \forall i, \quad \forall y$$

We can now solve the resulting program using standard numerical techniques. Alternatively, since the program is convex, we can equivalently solve the dual [12]:

$$\max_\alpha \sum_{i,y} \alpha_i(y)(1 - \delta_{y,y^i}) - \frac{1}{2}\|\sum_{i,y}\alpha_i(y)(\phi(\mathbf{x}^i, y^i) - \phi(\mathbf{x}^i, y)\|^2 \tag{13}$$
$$\text{s.t.} \quad \sum_y \alpha_i(y) = C \quad \forall i, \quad \alpha_i(y) \geq 0 \quad \forall i, \quad \forall y,$$

where we use $\alpha := (\alpha_i(y))_{(i,y)}$ ((i,y) is a multi-index that we use only here).

After solving the dual, we can express the weight vector as (which, as the dual itself, can be derived by considering the KKT optimality conditions):

$$\mathbf{w} = \sum_{i,y} \alpha_i(y)\left(\phi(\mathbf{x}^i, y^i) - \phi(\mathbf{x}^i, y)\right). \tag{14}$$

An advantage of using the hinge loss is, that in contrast to the solutions of the logistic regression objective, the solutions of this dual program are usually *sparse* wrt. α, ie. only a small number of $\alpha_i(y)$ are not equal to zero. This is an indirect result of using the non-smooth hinge-loss and the main motivation for considering this loss function as an alternative to the log-loss. Data-points $\mathbf{x}^i$ for which $\alpha_i(y^i) \neq C$ are called support vectors, and the overall model is called correspondingly **support vector machine** (SVM). Note that the dual is a problem in NK variables and constraints. For a reasonably small number of data-points and class labels the program can be solved efficiently. In Section 3.3.1, we will consider the problem for a huge number of class labels. As in the corresponding extension of logistic regression, specialized procedures will be required to solve the problem. If the number of points (and classes) is small, Problem 13 can solved by using standard techniques for solving convex quadratic programs. However, much more efficient methods have been suggested that make use of the special structure of the problem. A common procedure, 'sequential minimal optimization' (SMO) [53], solves the dual by successively solving for pairs of variables $\alpha_i(y), \alpha_j(y')$, which can be achieved in closed form.

The feature vectors appear in both, the dual program, and the application of the dual representation (Eq. 14) to test data, in the form of inner products only. We can replace these by positive definite, symmetric kernel functions and fit the model in a kernel induced feature space [60]. We discuss the use of kernels in more detail in the next section.

Note that the hinge-loss (Eq. 11) penalizes not only wrong classification decisions, but also *correct* decisions with low confidence. It is intuitively obvious, that minimizing it therefore gives rise to classifiers that are in a way more robust than those that we would obtain from minimizing the 0/1-loss. In the binary and linearly separable case this property is accompanied by the appealing intuition of *margins* [60]: By minimizing the regularized hinge-loss for a linearly separable, binary training set, we maximize the distance that the separating hyperplane has from the closest examples within each class. Note however, that the log-loss (Eq. 9), even though it does not share this intuition, also penalizes low confidence and leads to comparable generalization performance in practice. And similar to hinge-loss based classifiers, logistic regression can also be 'kernelized'. The only practically relevant differences between these two loss functions are therefore that the log-loss allows us to define a classifier probabilistically, while the hinge-loss leads to sparse solutions, and that parameter estimation usually involves unconstrained, gradient based, optimization for the first and the solution of a quadratic program for the latter.

Perceptron learning: An alternative algorithm for training a classifier of the form in Eq. 1 is the perceptron learning rule [58]. It does not require the solution of a constrained program and in contrast to logistic regression does not rely on a probabilistic model. A simple online algorithm that is based on the perceptron learning rule works as follows: First, we initialize the parameter vector to $\mathbf{w}_0 = 0$. We then sweep over the training data set T times (where T can be set beforehand), and in t^{th} sweep over the training data we compute $\hat{y} = \arg\max_y \mathbf{w}_t^T \phi(\mathbf{x}^i, y^i)$ and update the weight vector according to:

$$\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + \phi(\mathbf{x}, y^i) - \phi(\mathbf{x}, \hat{y}), & \text{if } \hat{y} \neq y^i, \\ \mathbf{w}_t, & \text{otherwise.} \end{cases} \tag{15}$$

The solution is then given by $\mathbf{w}_T$. Variations of this procedure are possible. One variation that can greatly improve generalization [21] consists in accumulating the parameter vectors during the sweeps over the training data and defining the final parameters as the average over these. For perceptron learning, similar to logistic regression and SVMs, theoretical guarantees exist regarding both convergence and generalization.

2.1.4 Kernels

An important property of linear classifiers is that they allow us to use the 'kernel trick' [60] and to fit the models in kernel induced feature spaces. A way of deriving a kernel based classifier is to replace the parametric definition of the compatibility measure $c(\mathbf{x}, y; \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}, y)$, by a specific non-parametric one: We define $c(\mathbf{x}, y)$ as an element of a Reproducing Kernel Hilbert Space [60] corresponding to a positive definite and symmetric *joint* kernel function $k((\mathbf{x}, y), (\mathbf{x}', y'))$. By the representer theorem [33] (see [28] or [6] for an adaption to the more general multi-class setting we discuss here), the solution of

Problem 5 under this definition admits a representation of the form:

$$c(\mathbf{x}, y) = \sum_j \sum_{y'} \alpha_j(y') k((\mathbf{x}, y), (\mathbf{x}^j, y')). \tag{16}$$

(Note that the variables $\alpha_i(y')$ are not the same as in the previous section.)

In contrast to binary settings, where we obtain a parameter for each data-point (see e.g. [60]), the multi-class setting yields a parameter $\alpha_i(y)$ for each data-point, class-label pair. The reason is that the loss-functions in multi-class settings generally depend on all labellings for each data-point (see e.g. Eqs. 9 and 11). (The adaptation of the representer theorem from the corresponding binary case to this case is straightforward.) We can now plug the dual representation of the compatibility measure (Eq. 16) back into the optimization problem and decision function and express these solely in terms of kernels. Optimization can then be performed with respect to the dual parameters $\alpha_i(y)$.

Note that in the previous section the possibility of using kernels followed already from the fact that in the dual program the feature vectors appeared only in the form of inner products. By invoking the representer theorem, however, we can derive kernel based variants for methods that use other loss-functions than the hinge-loss, and that do not rely on constrained optimization. By using the log-loss, for example, we obtain **kernel logistic regression**, by using other loss-functions we obtain other kernel based classifiers.

As mentioned previously, the parameters $\alpha_i(y)$ in Eq. 16 differ from those used in Eq. 14. Their relation can be derived straightforwardly, however: By using a simple calculation, we can rewrite Eq. 14 as:

$$\mathbf{w} = \sum_{j,y} [C\delta_{y,y^j} - \alpha_j(y)]\phi(\mathbf{x}^j, y). \tag{17}$$

Plugging this representation into the compatibility $\mathbf{w}^T\phi(\mathbf{x}, y)$ then shows that we can rewrite the representation in Eq. 16 using the coefficients $(C\delta_{y,y^j} - \alpha_j(y))$ where $\alpha_j(y)$ are the dual variables in the constrained program. We will discuss the use of kernels in more detail in the context of structured models in Section 3.4.

2.2 Unsupervised Inference

In this section we describe methods for defining and performing inference in complex, structured models. We consider models that capture global interactions between all quantities involved, but that for tractability express these in terms of only local interactions.

2.2.1 Graphical models and belief propagation

We can formalize the problem as follows. We define the vector $\mathbf{z}$, whose components contain all variables under consideration for a task at hand. The components can take on real values or discrete values, or be mixed. At times, it can be useful to distinguish between input variables $\mathbf{x}$ and output variables $\mathbf{y}$. To define a model we consider a scalar-valued function on $\mathbf{z}$. Such a global function could, for example, measure the goodness, or the probability, of joint configurations of $\mathbf{z}$. The inference task in such a model can then be defined as performing a global operation on $\mathbf{z}$ based on this function. Such a global operation could for example be a marginalization in the probabilistic case, or computing the best global configuration $\mathbf{z}$ in the 'goodness' case. Alluding to the probabilistic case, we will refer to any such global operation as 'marginalization'. To learn a model from training data we can parameterize it using some parameter vector $\mathbf{w}$ and maximize average 'goodness' or probability of the training set wrt. to $\mathbf{w}$. It can be useful to include 'hidden' variables in the model that are not instantiated in the training data. These can be dealt with conveniently in this setting by marginalizing them out in training. We consider the problem of parameter estimation in more detail in Section 2.2.3 and focus on inference in the following.

A general problem that we have to deal with when using a *global* function on $\mathbf{z}$ (such as a probability distribution), is the combinatorial explosion that arises from the exponentially many possible configurations of $\mathbf{z}$. A common solution to this tractability issue is to express the single global function in terms of several (also scalar valued) *local* functions that are defined only on subsets of the components of $\mathbf{z}$.

We will stack these components into sub-vectors $\mathbf{z}_s$ in the following. Computations that involve the joint assignment of values to $\mathbf{z}$ can then, under specific circumstances, be broken down into several tractable local computations. A formalism that allows for a unified treatment of such decompositions is given by graphical models [31]. A graphical representation that unifies several previous ones and that is useful for our purposes is given by **factor graphs** [35].

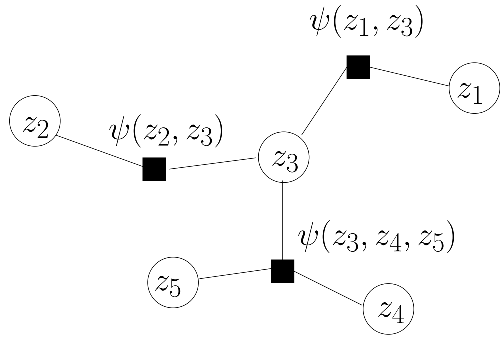

Figure 1: A factor graph.

A factor graph is an undirected graph, in which each variable (component of $\mathbf{z}$) and each local function is represented by a node. The only edges that are present are those that connect function nodes with their arguments. (Consequently, a factor graph is a bi-partite graph). An example of a factor graph is depicted in figure 1. The graph contains five variables and three local functions; the latter are denoted by $\Psi(\cdot)$. To obtain a global model from the local structure defined by the factor graph we have to choose an operation that combines the local functions into a single global function on $\mathbf{z}$. In a probabilistic model, the local functions can be local conditional distributions or 'potential functions', and the combining operation multiplication. Other operations can be used in other cases, and we provide examples below. The crucial point now is the following: If the nodes are arranged in a *tree* and if the operation used for combining the local functions, together with the operation used for marginalization, forms a **semi-ring**, then we can, by invoking a generalized form of the distributive law, perform inference efficiently[2] [34]: The number of required operations to perform marginalizations in the model scales polynomially with the number of joint configurations of the variables in the largest clique. We illustrate this point by means of an example.

Example (Markov random field): The Markov random field (MRF), also known as undirected graphical model [31], is an example of a probabilistic model that we can represent in the factor graph formalism. The MRF formalism is quite general and many models can be expressed in it, including Boltzmann machines [26] or products of experts [27]. An MRF is defined as the normalized product of potential functions, each of which is a positive, but otherwise arbitrary, function on a subset of the nodes of $\mathbf{z}$:

$$p(\mathbf{z}; \mathbf{w}) = \frac{1}{Z(\mathbf{w})} \prod_{s=1}^{S} \Psi(\mathbf{z}_s; \mathbf{w}), \tag{18}$$

$$Z(\mathbf{w}) = \sum_{\mathbf{z}} \prod_{s=1}^{S} \Psi(\mathbf{z}_s, \mathbf{w}) \tag{19}$$

(Note that we overload the symbol Ψ and let the argument determine which function is meant.) The potential functions are used to measure the desirability of joint instantiations of the variables within each clique, and the partition function $Z(\mathbf{w})$ to ensure normalization. Equation 18 can then be thought of as a way of designing a distribution that reflects these desirabilities by giving high probability to configurations that achieve high overall scores. Defining the joint as the normalized product amounts also to making conditional independence assumptions about the joint distribution. These independence assumptions are commonly referred as 'Markov' properties. Note that the operation to define the joint is multiplication, while the standard operation to perform inference in a probabilistic model is summation (to compute marginals or $Z(\mathbf{w})$, e.g.). The two operations form a semi-ring, and so inference can be performed efficiently. To compute, for example, the marginal distribution for variable z_1 under the model we have to compute a sum over all variables except z_1:

$$p(z_1; \mathbf{w}) = \frac{1}{Z(\mathbf{w})} \sum_{z_2} \cdots \sum_{z_I} \prod_{s=1}^{S} \Psi(\mathbf{z}_s; \mathbf{w}). \tag{20}$$

Now, by invoking the distributive law we can 'push' some of the summations into the product and compute the marginal by using a cascade of sums over products (ie. we evaluate Eq. 20 incrementally from 'right to left'). The largest number of values that we need to sum over at any instance in time then corresponds to the largest number of arguments that a single potential function references. Analogously, we can efficiently compute other marginals under the model, $Z(\mathbf{w})$, or expectations under the model. Notation can be simplified at times for these kinds of procedure by defining the 'not-sum' $\sum_{\sim z_i}$ as a shorthand for $\sum_{z_1,\ldots,z_{i-1},z_{i+1},\ldots,z_I}$, which is simply the sum over all component except for z_i. We will make use of this notation later on in this review.

A common alternative to the probabilistic 'sum-product' semi-ring is the 'min-sum' semi-ring. We obtain this semi-ring by defining a global function on $\mathbf{z}$ as the sum of local constraints or 'costs' associated with subsets of the variables and treating inference as the problem of minimizing the overall cost. (These two ways – 'sum-product' vs. 'min-sum' – of constructing a model are intimately related: While the first corresponds to a probabilistic model, we can obtain the latter from the first by considering minimizations in the model, and performing it in the (negative) log-domain.)

For models whose underlying factor graph is a *tree*, it is possible to generalize the efficient computation of one single node marginal to obtain *all* single-node marginals in the graph ([35], [31]). The basic insight is that we can interpret the resulting series of local summations and products as **messages** that are passed between nodes of the graph, which allows us to reuse summations when computing marginals. Since a factor graph contains two kinds of variable, there are two kinds of message we have to consider: Those that are sent from variable- to function-nodes and those that are sent from function- to variable-nodes. The first are usually denoted $\mu_{z\to\Psi_s}$, where z is a variable node and Ψ_s a function node; the latter are denoted $\mu_{\Psi_s\to z}$. Formally, using the 'sum-product' semi-ring as an example, we obtain messages of the form [35]:

$$\mu_{z\to\Psi_s}(z) = \prod_{h\in n(z)\setminus\{\Psi_s\}} \mu_{h\to z}(z), \tag{21}$$

$$\mu_{\Psi_s\to z}(z) = \sum_{z'\neq z} \left(\Psi_s(\mathbf{z}_s) \prod_{z'\in n(\Psi_s)\setminus\{z\}} \mu_{z'\to\Psi_s}(z') \right), \tag{22}$$

where $n(z)$ denotes the set of nodes that are connected to z by an edge, ie. the 'neighbors' of node z, and $\mathbf{z}_s$ the vector of variables connected to Ψ_s. Variables that are connected to only one function node send the trivial message 1. In practice not necessarily all variables are marginalized over. In particular, some variables can be observed, ie. they can be fixed to specific values. The messages for these variables are computed as above for the observed values, and are set to 0 for all other values that the variable can take on. Once messages have been passed along every direction on every edge of the graph, which is trivially possible if the graph is a tree, marginals can be computed for each single node simply as the product of incoming messages.

A special case of probabilistic models is given by models whose potential functions represent local conditional probabilities. Similar to undirected models (Eq. 18) they define the global distribution as the product of local functions, but they are in contrast to those models automatically normalized, and therefore do not require the normalization constant. The models can be viewed as defining the global distribution by means of local causal relationships. They are commonly referred to **directed models**, because these causal relations can, alternatively to the factor graph representation, be represented using directed (acyclic) graphs.

Example (Hidden Markov Model): Hidden Markov Models (HMMs) are a standard example for directed models. They define the joint distribution over *sequences* $\mathbf{z} := (\mathbf{x}, \mathbf{y}) := (x_1, \ldots, x_T, y_1, \ldots, y_T)$

as:

$$p(\mathbf{x}, \mathbf{y}) = \prod_t p(y_t|y_{t-1})p(x_t|y_t). \tag{23}$$

The standard inference problem for HMMs consists in computing single-node marginal probabilities $p(y_t|\mathbf{x})$ given an observation sequence $\mathbf{x}$. Applying the message updates (Eqs. 21 and 22) in this model recovers the well-known α- and β-recursions normally used for this purpose [56]. A related problem is Viterbi decoding, which consists in computing $\arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. It is usually solved in the log-domain by using the operations from the corresponding 'max-sum'-semi-ring.

When the local distributions are modeled independently of the 'time'-index t, the HMM is called *stationary*, and can for discrete $\mathbf{y}$ be thought of as a probabilistic finite state machine. We can think of stationary models also in terms of parameter-tying. We will discuss this issue in more detail in the context of structured response models in Section 3.1.2.

2.2.2 Approximate inference

If the graph is not a tree, we cannot apply belief propagation to perform inference efficiently. In probabilistic models we can efficiently perform *approximate inference* instead. A common approach consists in replacing the intractable distribution $p(\mathbf{z})$ by a tractable surrogate distribution $q(\mathbf{z})$ and performing inference using this distribution. Besides allowing for tractable inference, the surrogate distribution needs to be 'close' to the true distribution. To meet both requirements, often a fully factorized or tree-structured distribution is chosen, that has small Kullback-Leibler divergence $\mathrm{KL}(q(\mathbf{z}), p(\mathbf{z}))$ to the true distribution (see e.g. [22] for a tutorial overview).

An alternative approach to performing approximate inference consists in ignoring the fact that the graph is not a tree and running belief propagation nevertheless, by repeating local message updates according to some pre-defined schedule. In some cases the process will converge and yield consistent marginal distributions. Another alternative approach consists in approximating the intractable distribution by a representative sample drawn from it, which can then be used to approximate, for example, expectations under the distribution (see [49]).

2.2.3 Parameter estimation

Usually, a model is defined up to a set of free parameters, that need to be estimated from training data. The standard approach for training *probabilistic* models (we restrict our attention to these in the following) consists in maximizing the log-likelihood of the training data:

$$\mathcal{L}(\mathbf{w}) = \sum_i \log p(\mathbf{z}^i; \mathbf{w}) \tag{24}$$

wrt. $\mathbf{w}$. A (local) maximum of the log-likelihood could be found by using, for example, gradient based optimization wrt. $\mathbf{w}$. (Note that for discrete models we can often parameterize the potential functions directly, instead of using the parameterization $\mathbf{w}$. In that case, often closed-form solutions exist, that are based on simple instance counting.)

As mentioned above, it is common to leave some variables uninstantiated during training, either because not all values are available for each training case, or because we deliberately equip the model with hidden variables that can pick up structure during training. In the latter case, these variables are typically uninstantiated across all elements of the training set. The standard approach to estimating parameters in the presence of hidden variables $\mathbf{h}$ is maximization of the *marginal log-likelihood* of the observable data $\mathbf{v}$ (we assume $\mathbf{z} = (\mathbf{h}, \mathbf{v})$):

$$\mathcal{L}^o(\mathbf{w}) = \sum_i \log \sum_{\mathbf{h}^i} p(\mathbf{h}^i, \mathbf{v}^i; \mathbf{w}). \tag{25}$$

The marginal likelihood could be maximized either directly, or by using the (generalized) EM-algorithm [50], which can help subdivide the optimization problem into a sequence of more efficiently solvable subproblems. Using the EM-algorithm requires repeatedly performing inference in the model, for which any of the methods discussed above can be deployed.

Approximate parameter estimation: For a large class of models, computing the gradient requires evaluating an expectation under the distribution defined by the model. For intractable models, we can *approximate* the expectations using any of the approaches discussed in Section 2.2.2. We can also derive approximation methods specifically for the purpose of parameter estimation. An efficient sampling approach for parameter estimation, for example, that has been applied also in the structured models that we describe in the following section, is contrastive divergence [27].

Fully Bayesian models: An alternative way to perform parameter estimation is to treat the parameters as variables themselves. Parameter estimation then becomes the problem of determining the posterior distribution $p(\mathbf{w}|\mathbf{y}, \mathbf{x})$, which is just an inference problem in the resulting overall model. In principle, a fully Bayesian model performs always either as well as, or better than, a model that is not Bayesian wrt. the parameters (note, that we can view the latter as special case of a fully Bayesian model, where we perform approximate inference using a delta-distribution as the approximating distribution). However, in practice, including the parameters into the model usually leads to extra intractabilities and can make learning difficult.

3 Structured response models

In this section we discuss the problem of constructing complex models by combining discriminative and unsupervised approaches. The underlying modeling philosophy consists in using supervised methods to perform parameter estimation for complex models, in which inference can then be performed by using the standard techniques from unsupervised learning.

3.1 Cost functions and inference

We consider the problem of predicting structured outputs in a discriminative setting. Several methods (see e.g. [37], [65], [13]) have been proposed for this purpose in the last few years. But many of them turn out to share several core ideas, and so a general framework for the problem of predicting structured outputs has emerged.

The central idea, which is common to the different methods, is to treat the issue as a *linear classification* problem. We are then looking for a predictor of a form similar as in Section 2.1, with the difference that the system output is no longer a scalar class label, but an arbitrary structured object. Formally, we can represent the output using the discrete vector $\mathbf{y} := (y_1, \ldots, y_T)^T$, and in analogy to Section 2.1.1 define a joint feature vector $\phi(\mathbf{x}, \mathbf{y})$. We can then consider classifiers based on the score $c(\mathbf{x}, \mathbf{y}; \mathbf{w}) = \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$. A *joint* feature representation is obviously useful in this setting, where not only the inputs, but also the outputs can be structured objects.

The fact that the output $\mathbf{y}$ is a vector as opposed to a simple scalar class label distinguishes structured response models from simple classification methods in three important ways. The first difference concerns the definition of the decision and risk functions. A simple adaptation of the single output predictor (compare to Eq. 1) to deal with vector-valued outputs:

$$\mathbf{f}(\mathbf{x}) = \arg\max_{\mathbf{y}} \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y}), \tag{26}$$

is not the only reasonable, and not necessarily the best, choice for all situations. When the labels are not single homogeneous objects, decision criteria other than those that maximize the overall score on the whole structure to be predicted, are imaginable. One alternative criterion, for example, would be to correctly predict as many *single* entries of $\mathbf{y}$ as possible. A related issue concerns the loss function chosen for training. For estimating parameters of a structured response model the direct generalizations (Eqs. 9 and 11) of the 0/1-loss are not necessarily the most suitable and it might be more natural to replace these by loss functions that depend on the number of misclassified single labels ('Hamming loss'). The way that these alternative loss functions can be incorporated differ greatly across different models. Margin based approaches incorporate these implicitly by modifying the constraints of the constrained optimization problem on which they are based. Probabilistic models provide more direct ways of using non-standard loss functions by making it possible to optimize over marginal distributions. Details on how

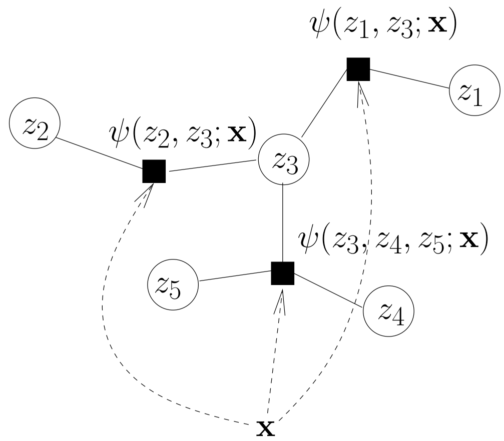

Figure 2: Factor graph representation of a structured response model.

to incorporate different kinds of loss function will be discussed in Sections 3.2.1 and 3.3.1 as part of the general parameter estimation problem.

The second difference to standard classification problems is that, regardless of the chosen loss function, inferring a structured response using Eq. 26 is often much harder than inferring a single class label using Eq. 1. The reason is that $\mathbf{y}$ can take on one of a potentially huge number of values, and so we cannot simply plug in each possible value to determe the maximum. Instead dynamic programming or other inference approaches from Section 2.2 can be used. The adaptation of these to structured response models is straightforward as will be described in the next section.

The third difference is that parameter estimation for structured response models is more difficult than for standard classification and often requires special insights. In principle, all parameter estimation techniques (such as gradient based or constrained optimization, or the perceptron learning rule) could be transferred directly from the standard cases described in Section 2.1. However, for structured response models, these methods pose extra challenges, that we will discuss, along with the issue of loss functions, in Sections 3.2.1 and 3.3.1.

3.1.1 Inference and decomposable feature functions

Performing inference, using for example Eq. 26, is difficult for the same reasons as in generative models. However, similarly as in section 2.2, we can solve the problem here by making use of suitable decompositions of the feature vector $\phi(\mathbf{x}, \mathbf{y})$. In particular, we can decompose $\phi(\mathbf{x}, \mathbf{y})$ into sub-vectors $\phi(\mathbf{x}, \mathbf{y}_s)$, $s = 1, \ldots, S$, each of which depends only on the subset of the components of $\mathbf{y}$ – the components that belong to a particular clique s of an underlying graph. Using a corresponding decomposition for $\mathbf{w} = (\mathbf{w}_s)_{s=1,\ldots,S}$ we can write Eq. 26 as:

$$\mathbf{f}(\mathbf{x}) = \arg\max_{\mathbf{y}} \sum_s \mathbf{w}_s^T \phi(\mathbf{x}, \mathbf{y}_s). \tag{27}$$

Now the 'max' distributes over the sum and so belief propagation can be used to compute it efficiently.

We can represent the resulting model as a '*conditional factor graph*' as illustrated in figure 2. The local functions are $\psi(\mathbf{y}_s; \mathbf{w}, \mathbf{x}) = \mathbf{w}_s^T \phi(\mathbf{x}, \mathbf{y}_s)$. The dependencies of the feature functions on the inputs are indicated with dashed arrows. Instead of incorporating the observations into the model, we use the model only to define the *response* of the system to an input $\mathbf{x}$. In order to make that response a function of the input, we parameterize the function nodes *using the inputs themselves as parameters.* In Section 3.2 we will consider the probabilistic analog to Eq. 26, that involves computing expectations instead of the 'arg-max', and is based on the 'sum-product' semi-ring. As in generative models, we can also use a graph structure that is not a tree and then resort to variational methods to perform inference.

Note that the dependency of $\phi(\mathbf{x}, \mathbf{y})$ on the *input* $\mathbf{x}$ can be chosen in an arbitrary way. In particular, a decomposition with respect to the input is not necessary, because the 'arg-max' is over only $\mathbf{y}$. This is a direct consequence of the fact that the input is never itself modeled, and it is in contrast to generative models, such as HMMs, that define a *joint* model over data and system variables.

Note that the actual reason for tractability is not linearity of the model (Eq. 26), but rather *additivity*. We could therefore replace the local linear functions $\mathbf{w}_s$ by arbitrary nonlinear functions, such as neural networks. Currently, models based on linear scores are much more common than models based on nonlinear scores, but this might change, because the use of nonlinear scores could provide a great advantage in terms of accuracy.

3.1.2 Chains, lattices and parameter tying

As in generative approaches, an important subclass of structured response models are chain structured models. They can be defined by arranging the components of $\mathbf{y}$ in a chain and by using features that reference only adjacent nodes. The most common choice is to define features $\phi_s(\mathbf{x}, \mathbf{y}_{s-1}, \mathbf{y}_s)$ on pairs of nodes. Such a choice may be viewed as the conditional analog to the first order Markov assumption made in HMMs. Another important subclass, that is particularly common in image processing tasks, are models whose nodes are arranged on a regular lattice. Features for these models can be naturally defined by using pair-wise cliques, each of which is defined on a node and its neighbors. In image processing tasks these models can be used to capture local consistency constraints on pixels, for example.

A common design principle for both, sequence and lattice structured, models is based on **templates**. In analogy to the constant transition probabilities used in stationary HMMs, we can base a structured response model on feature functions that are *shared* between cliques. Since the features are parameterized by $\mathbf{w}$, using the same feature function corresponds to tying the parameter vector across cliques, ie. to setting $\mathbf{w}_s = \mathbf{w}$ for all s. By linearity, Eq. 27 then simplifies to

$$\mathbf{f}(\mathbf{x}) = \arg\max_{\mathbf{y}} \mathbf{w}^T (\sum_s \phi(\mathbf{x}, \mathbf{y}_s)). \tag{28}$$

In other words, we obtain a global feature vector by summing over the feature vectors evaluated at the single cliques. To define a model, therefore all we need to do is define the template of a feature vector on a single clique. Besides reducing the number of parameters, this approach provides a simple way of dealing with models of variable sizes. However, it is valid only under the assumption that decisions on every clique actually do depend on the same features, ie. that all cliques are equivalent in this strong sense. Note that combinations of a fully-parameterized and a template-based modeling approach are possible in principle.

[63] discuss a template based approach in the context of complex sequence structured models, in which each 'time slice' contains several nodes, that can be structured in complicated ways. Complex structures can easily make tractable inference impossible, and so in general approximate inference needs to be used in these models.

3.2 Probabilistic models

We now turn to the problem of defining an actual trainable model, that implements the cost function and decomposable features described in the previous section. We focus on probabilistic approaches in this section and discuss non-probabilistic ones in Section 3.3. Both kinds of approach can be viewed as direct generalizations of the methods discussed in Section 2.1, ie. logistic regression on the one hand and margin based methods on the other. However, the parameter estimation problems differ from the corresponding problems for the standard methods. In particular, they pose extra challenges, such as an exponential number of constraints for the non-probabilistic methods, and intractable model expectations for the probabilistic ones.

We can obtain a probabilistic structured response model simply by generalizing the logistic regression model (Section 2.1.2) to vector valued outputs. In direct analogy to Eqs. 6 and 7 we can define:

$$p(\mathbf{y}|\mathbf{x};\mathbf{w}) = \frac{1}{Z(\mathbf{x};\mathbf{w})} \exp(\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})), \quad \text{with} \tag{29}$$

$$Z(\mathbf{x};\mathbf{w}) \quad = \quad \sum_{\mathbf{y}} \exp(\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})). \tag{30}$$

Now, if the feature functions decompose according to Eq. 27, we have:

$$p(\mathbf{y}|\mathbf{x};\mathbf{w}) = \frac{1}{Z(\mathbf{x};\mathbf{w})} \prod_s \exp(\mathbf{w}_s^T \phi_s(\mathbf{x},\mathbf{y}_s)). \tag{31}$$

We obtain a Markov random field conditional on the input $\mathbf{x}$. The (input dependent) potential functions are given by $\Psi(\mathbf{y}_s) := \exp(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y}_s))$. Training this model is equivalent to choosing a function that maps an input vector $\mathbf{x}$ to an MRF over $\mathbf{y}$. The inference problem is exactly the same as in a conventional MRF: At 'test time', an input $\mathbf{x}^{\text{test}}$ simply induces an MRF over $\mathbf{y}$, and we can deploy any of the techniques described in Section 2.2 to perform inference. Since the model is probabilistic, we can consider inference problems other than Eq. 27, such as the computation of single-node marginals or drawing samples from the resulting distribution, for example.

A simple early version of this idea has been described in [1] as an input-output version of a Boltzmann machine. [40] discuss a general framework for combining dynamic-program type inference with gradient based learning. They do not restrict their attention to probabilistic models and consider more general, non-linear models. The special case of a linear model has been made popular only much later as a **conditional random field** (CRF) by [37], who suggested using this kind of model for labelling sequences. CRFs have since then been adapted and extended in several ways (see e.g. [63], [59]) and applied to several problems in natural language (e.g. [61], [43]) and image processing (e.g. [55], [25]), among others.

While CRFs can be viewed as conditional undirected graphical models, conditional *directed* models have also been suggested. One example is the Maximum Entropy Markov Model (MEMM) [44]. A MEMM is a Markov model whose state transition probabilities are defined as (input dependent) maximum entropy distributions. When used with parameter tying, the model can be viewed as a (conditional) finite state machine. [37] point out a potential problem of such models, that they call 'label bias' and whose discovery they attribute to [11]. The flaw is due to the local normalization, which causes state transitions to compete only against other transitions from the same state and not against all other transitions in the model. In a model that is not fully connected, ie. in which some state transition probabilities are zero, decoding will tend to favor states with a small number of outgoing transitions – independently of the input. One motivation for the introduction of CRFs has been to overcome this problem through the use of global normalization (ie. an undirected model). The effects of the label bias problem have been verified in some experiments (e.g. [37]), and have somewhat reduced the interest in those models in the last few years.

Several earlier methods can also be viewed as directed conditional models. One example is the input-output HMM (IOHMM), introduced by [9], who define the transition probabilities using neural networks. We can also view the classical mixture of experts model [30] as a directed conditional model (with a degenerately simply 'structured' response), since its optimization criterion is conditional likelihood. IOHMMs and mixtures of experts differ from the other conditional models discussed so far, in that they also contain *hidden variables*, that is variables that are not instantiated in the training data. The EM algorithm is commonly used to train these models. We will discuss the use of hidden variables in structured response models in more detail below.

Example (Linear chain CRF): Of foremost interest in practical applications have been sequence structured CRFs ([37], [63]), and in particular those whose feature vectors are shared across cliques. For models that are first order Markov, the features reference pairs of nodes and are of the form: $\phi(\mathbf{x}, y_t, y_{t-1})$. The standard inference problem for these models is similar as in HMMs and can be derived from belief propagation on the conditional distribution. Performing belief propagation results in recursive update equations similar to the 'forward-backward' equations. Defining special 'start' and 'stop' states for the model, we can write single node marginals compactly [37] as (note that for consistency with the literature, we use variables α here, even though we have used the same variable in a different context in Section 2.1.3 and will use them again in that context later in this review.):

$$p(y_t = \hat{y}|\mathbf{x};\mathbf{w}) = \frac{\alpha_t(\hat{y};\mathbf{x},\mathbf{w})\beta_t(\hat{y};\mathbf{x},\mathbf{w})}{Z(\mathbf{x};\mathbf{w})}, \tag{32}$$

where we use the recursive definitions

$$\alpha_0(\hat{y};\mathbf{x},\mathbf{w}) = \delta_{\hat{y},\text{start}} \qquad \alpha_t(\mathbf{x},\mathbf{w}) = \alpha_{t-1}M_t(\mathbf{x},\mathbf{w}) \tag{33}$$

$$\beta_{T+1}(\hat{y};\mathbf{x},\mathbf{w}) = \delta_{\hat{y},\text{stop}} \qquad \beta_t(\mathbf{x},\mathbf{w})^T = M_{t+1}(\mathbf{x},\mathbf{w})\beta_{t+1}(\mathbf{x},\mathbf{w}), \tag{34}$$

where α_t is a vector with one component for each (single-node) state y_t, and $M_t(\mathbf{x},\mathbf{w})$ a matrix with entries

$$M_t(y_t, y_{t-1};\mathbf{x},\mathbf{w}) = \exp(\mathbf{w}^T\phi(\mathbf{x}, y_t, y_{t-1})), \tag{35}$$

and the partition function $Z(\mathbf{x};\mathbf{w})$ can be computed as

$$Z(\mathbf{x};\mathbf{w}) = \left(\prod_{t=1}^{T+1} M_t(\mathbf{x},\mathbf{w})\right)_{\text{start,stop}} \tag{36}$$

Instead of using decompositions to obtain tractable models, we can alternatively use intractable models and resort to variational methods or sampling to perform approximate inference. [63] for example, discuss the use of loopy belief propagation on a natural language processing task. An example of an intractable conditional model applied in an image processing task is given in [25]. The conditional model is a restricted Boltzmann machine and contrastive divergence is used for parameter estimation.

3.2.1 Parameter estimation

The close connection of CRFs to logistic regression makes parameter estimation straightforward. We can basically adopt the procedure described in Section 2.1.2. Two points that complicate the problem when dealing with structured outputs have to be pointed out, however: The first is, that parameter estimation usually involves computing expectations under the model distribution. We therefore have to make use of either the feature decomposition (Eq. 27) or approximate inference (Section 2.2.2) to perform parameter estimation efficiently. The second point is, that the choice of loss function deserves special attention in this setting, since in the presence of multiple labels a simple distinction between wrong and correct labels is not necessarily the most appropriate choice. We will discuss the choice of loss function in detail in the following, and turn to actual procedures for parameter estimation below.

Loss functions: The most straightforward way to train a CRF is to proceed as in Section 2.1.2 and maximize (a regularized version of) the log-likelihood

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_i \log p(\mathbf{y}^i|\mathbf{x}^i,\mathbf{w}) \tag{37}$$

$$= \frac{1}{N}\sum_i \left[\mathbf{w}^T\phi(\mathbf{x}^i,\mathbf{y}^i) - \log Z(\mathbf{x}^i,\mathbf{w})\right]. \tag{38}$$

As before, we are implicitly using the log-loss (Eq. 9) as our loss function, when following this approach. Recall that the log-loss compares the confidence of the model in the true label to the confidences in all other labels. However, in contrast to simple multi-class classification, here a label is a structured object, and such a comparison is therefore not necessarily the only reasonable choice on which to base a loss function. Instead, we might consider loss functions that compare for example single components or subsets of the components of $\mathbf{y}$. A simple loss-function that realizes a single-component comparison has been suggested by [32]. It is defined as the *average single-node log-likelihood*:

$$\mathcal{L}^{\text{avg}}(\mathbf{w}) = \frac{1}{SN}\sum_i\sum_s \log p(y_s^i|\mathbf{x}^i,\mathbf{w}). \tag{39}$$

Note that training a model with this loss-function optimizes its accuracy on single labels. Therefore, to apply such a model at test time, we have to classify single labels according to their marginal probabilities instead of labelling a whole sequence based on its joint probability. [32] report on improvements using this strategy on several standard datasets. Currently, Eq. 37 is still found most often in practice.

As in Section 2.1.2, to regularize, a penalty term might be added to the loss function. A full Bayesian approach is possible too, but usually requires variational approximations to be tractable. See [54] for a Bayesian CRF based on expectation propagation. In the following we focus on regularized maximum likelihood. Despite convexity of the objective function, parameter estimation is in general not easy and involves iterative procedures. Several approaches have been suggested for maximizing the regularized log-likelihood and we consider the most common ones in the following:

Iterative scaling: In their original paper, [37] suggest using a variant of improved iterative scaling (IIS) [19] to find the maximizer of (37). Iterative scaling approaches have a longstanding history as parameter estimation methods for maximum entropy or logistic regression type of models. They are based on the iterative maximization of a lower bound of the objective function, which can be performed efficiently in a coordinate-wise fashion. Currently a strong trend away from iterative scaling approaches towards standard gradient based optimization methods is noticeable ([70], [61]), supported by, amongst others, two empirical comparisons of several optimization methods for these kinds of models ([47], [42]). We therefore do not discuss IIS in detail in the review and focus on the more common gradient based methods instead.

Gradient based methods: The most common optimization methods for CRFs are gradient based methods, because they have shown to be among the most efficient methods on several standard problems [70], [61]. Since a CRF is a kind of logistic regression model, gradient based optimization can be performed in a similar way as in Section 2.1.2. Similarly as before (Eq. 10), the gradient of the log-likelihood takes the form:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \sum_i \phi(\mathbf{x}^i, \mathbf{y}^i) - \mathrm{E}_{p(\mathbf{y}|\mathbf{x}^i;\mathbf{w})} \phi(\mathbf{x}^i, \mathbf{y}), \tag{40}$$

where in contrast to before computing expectations over the model distribution requires to sum over a potentially huge number of terms. In general, we therefore need to make use of special procedures to compute them efficiently. As in standard inference problems, the most common approach is to make use of the decomposition (Eqs. 27, 31) and to use belief propagation to compute the sum. The alternative, single-node objective (Eq. 39) can be dealt with analogously. For linear chain models, the detailed update equations are given in [32]. The use of variational approximations, instead of tractable decompositions, can be easily generalized from the corresponding unconditional models, too. See [25], for example, for an application of contrastive divergence to training an intractable CRF.

Second order information can in general greatly increase the efficiency of gradient based optimization methods [51]. For CRFs the use of second order information poses additional computational challenges, that are due to the form and the size of the Hessian:

$$\nabla^2 \mathcal{L}(\mathbf{w}) = \sum_i \left(E_{p(\mathbf{y}|\mathbf{x}^i;\mathbf{w})} \phi(\mathbf{x}^i, \mathbf{y}) \phi(\mathbf{x}^i, \mathbf{y})^T - E_{p(\mathbf{y}|\mathbf{x}^i;\mathbf{w})} \phi(\mathbf{x}^i, \mathbf{y}) E_{p(\mathbf{y}|\mathbf{x}^i;\mathbf{w})} \phi(\mathbf{x}^i, \mathbf{y})^T \right). \tag{41}$$

Since the size of the matrix is quadratic in the number of features, using a full Hessian is usually not feasible. Furthermore, its computation involves computing the expectation of an outer product of feature vectors. Correspondingly, each entry requires computing the expectation of a product of pair-wise features, which in turn amounts to considering the joint instantiations of all pairs of cliques. A natural way of dealing with both issues is to approximate the Hessian by discarding all off-diagonal terms.

However, additional special considerations are necessary, when using parameter-tying. In that case, discarding off-diagonal terms will not prevent the coupling of cliques within each entry, since each feature itself is the sum of clique-wise features (Eq. 28). A further approximation, that [61] suggest as part of a pre-conditioning strategy for early iterations of a conjugate gradient (CG) method, is to simply replace the square of the sum of clique-wise features by the sum of the squares of the features. While this approximation led to reportedly good results when used in the early CG-iterations, it is probably too crude to be useful in more general ways.

In general, the difficulties with computing the Hessian suggest that using quasi-Newton methods, such as L-BFGS, might be more suitable optimization approaches for CRFs. These kinds of method approximate the Hessian during optimization by accumulating information contained only in the gradient and objective function values. A general assessment of which method is the best one in practice is difficult, partly because the optimal method could be task-dependent. For a comparison of different optimization methods on several natural language processing tasks using linear-chain models see [61] and [70].

3.2.2 Hidden variables

As in generative models, we can consider hidden variables, ie. variables that are not instantiated during training. In contrast to generative models, where hidden variables usually capture structure that is hidden in the input data, in CRFs they could capture structure that is *useful* for the task at hand. Training in the presence of hidden variables could be performed similarly as in generative models, by considering the marginal likelihood of the observed output variables:

$$\mathcal{L}^o(\mathbf{w}) = \sum_i \log \sum_{\mathbf{h}^i} p(\mathbf{h}^i, \mathbf{y}^i|\mathbf{x}^i, \mathbf{w}), \tag{42}$$

which for CRFs (Eq. 29) expands to:

$$\mathcal{L}^o(\mathbf{w}) = \sum_i \left[\log \sum_{\mathbf{h}^i} \exp(\mathbf{w}^T \phi(\mathbf{h}^i, \mathbf{y}^i|\mathbf{x}^i) - \log \sum_{\mathbf{h}^i, \mathbf{y}^i} \exp(\mathbf{w}^T \phi(\mathbf{h}^i, \mathbf{y}^i|\mathbf{x}^i) \right]. \tag{43}$$

The gradient is given by

$$\frac{\partial \mathcal{L}^o(\mathbf{w})}{\partial \mathbf{w}} = \sum_i \left[E_{p(\mathbf{h}^i|\mathbf{x}^i,\mathbf{y}^i)} \phi(\mathbf{x}^i, \mathbf{h}^i, \mathbf{y}^i) - E_{p(\mathbf{h}^i,\mathbf{y}^i|\mathbf{x}^i)} \phi(\mathbf{x}^i, \mathbf{h}^i, \mathbf{y}^i) \right]. \tag{44}$$

The marginal likelihood can be optimized either directly or by using a version of the EM-algorithm. Note that as in generative models the marginal likelihood will in contrast to the fully observed likelihood not be convex in general, and so we are only guaranteed a local maximum.

The use of hidden variables in structured response models has been suggested and rudimentarily applied by [55], [9], [32], [25]. A general evaluation of the benefits as well as technical issues of using hidden variables in these models has to the best of our knowledge not been conducted yet and will require further research.

3.3 Direct models

As in standard classification models, we can consider other loss-functions than those derived from a probabilistic model. In this section we discuss the use of the hinge-loss, which gives rise to structured equivalents of margin based methods (Section 2.1.3). Similarly as before, the fact that the outputs are structured leads to computational challenges. But the challenges for these models take a different form than those for CRFs: As in Section 2.1.3 we will cast the parameter estimation problem as a constrained optimization problem. For structured problems, the number of constraints is exponential in the number of label sites and so the challenge consists in dealing with a quadratic program with a very large number of constraints.

3.3.1 Parameter estimation

For parameter estimation we can consider a quadratic program as in Section 2.1.3:

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i \xi_i \tag{45}$$

$$\text{s.t.} \quad \mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y}^i) - \mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y}) \geq 1 - \delta_{\mathbf{y},\mathbf{y}^i} - \xi_i, \quad \forall i, \quad \forall \mathbf{y} \tag{46}$$

As in the discussion of CRF loss functions, here the direct adaptation of the optimization problem from the standard classification setting treats all mis-classifications equally, which is not quite appropriate for structured outputs. Two possible modifications of the quadratic program have been suggested to deal with this issue. Both are based on modifications of the constraints in Problem 45. [65] suggest *rescaling the margin* using the loss function as a scaling factor. This can be achieved by using the alternative constraint:

$$\mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y}^i) - \mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y}) \geq \Delta(\mathbf{y}^i, \mathbf{y}) - \xi_i, \quad \forall i, \quad \forall \mathbf{y}, \tag{47}$$

where $\Delta(\mathbf{y}^i, \mathbf{y}) = \sum_{t=1}^T \delta_{y_t^i, y_t}$ counts the number of wrongly classified components y_t of $\mathbf{y}$. [68] suggest alternatively *rescaling the slack variables* using the inverse loss as a scaling factor. The constraints then take the form

$$\mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y}^i) - \mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y}) \geq 1 - \delta_{\mathbf{y}, \mathbf{y}^i} - \frac{\xi_i}{\Delta(\mathbf{y}^i, \mathbf{y})}, \quad \forall i, \quad \forall \mathbf{y}. \tag{48}$$

In a comparison on some standard task, [68] report similar performance across all three approaches. However, further research could be useful for gaining deeper insights into how the approaches compare.

As in Section 2.1.3 it is useful in practice to consider the dual program. Depending on the strategy chosen for incorporating the non-uniform loss (margin rescaling or slack rescaling), we obtain two slightly different dual programs (see e.g. [68]). They can both be derived straightforwardly using Lagrangian duality. For the latter case (slack rescaling) the dual takes the form:

$$\max_\alpha \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y})(1 - \delta_{y,y^i}) - \frac{1}{2} \| \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y})(\phi(\mathbf{x}^i, \mathbf{y}^i) - \phi(\mathbf{x}^i, \mathbf{y})) \|^2 \tag{49}$$

$$\text{s.t.} \quad \sum_{\mathbf{y}} \frac{\alpha_i(\mathbf{y})}{\Delta(\mathbf{y}^i, \mathbf{y})} = C, \quad \forall i, \quad \alpha_i(\mathbf{y}) \geq 0, \quad \forall i, \quad \forall \mathbf{y}.$$

Note the difference in the first constraint when comparing to the standard multi-class setting. For the margin rescaling approach the constraint remains unchanged, but the linear part of the objective function changes. We obtain the program:

$$\max_\alpha \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \Delta(\mathbf{y}^i, \mathbf{y}) - \frac{1}{2} \| \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y})(\phi(\mathbf{x}^i, \mathbf{y}^i) - \phi(\mathbf{x}^i, \mathbf{y})) \|^2 \tag{50}$$

$$\text{s.t.} \quad \sum_{\mathbf{y}} \alpha_i(\mathbf{y}) = C, \quad \forall i, \quad \alpha_i(\mathbf{y}) \geq 0, \quad \forall i, \quad \forall \mathbf{y}.$$

The dual representation of the weight vector, similarly as in Section 2.1.3 (Eq. 14) is given by:

$$\mathbf{w} = \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\phi(\mathbf{x}^i, \mathbf{y}^i) - \phi(\mathbf{x}^i, \mathbf{y}) \right). \tag{51}$$

In both the primal and dual programs the numbers of constraints scale linearly with the number of classes, which in turn scales exponentially with the number of labels. Specialized procedures are therefore necessary to solve them. Below we discuss two approaches that directly deal with the dual and another approach that uses perceptron learning to solve the problem indirectly. Note also that in the dual, the feature vectors appear only in inner products. We can therefore replace these by kernels and fit the models in corresponding kernel feature spaces. We will discuss the use of kernels in detail and more generality in Section 3.4.

Marginal variables: In the dual programs both, the number of variables and the number of constraints, is exponential in the number of labels. However, [65] show that it is possible to replace Problem 50 by an equivalent quadratic program that is only polynomial in the number of labels. A condition under which this is possible is that both the risk function $\Delta(\mathbf{y}^i, \mathbf{y})$ and the feature function $\phi(\mathbf{x}, \mathbf{y})$ decompose into the sum of clique-wise terms. The crucial observation is that $\alpha_i(\mathbf{y})$ can for each i be interpreted as a *density* over $\mathbf{y}$, because of the constraints $\sum_{\mathbf{y}} \alpha_i(\mathbf{y}) = C$ and $\alpha_i(\mathbf{y}) \geq 0$. Furthermore, the objective function (50) contains expectations of the loss- and the feature functions under this density. If the loss-function decomposes as

$$\Delta(\mathbf{y}, \mathbf{y}') = \sum_s \Delta(\mathbf{y}_s, \mathbf{y}'_s), \tag{52}$$

we can re-write these expectations as follows:

$$\sum_{\mathbf{y}} \alpha_i(\mathbf{y}) \Delta(\mathbf{y}^i, \mathbf{y}) \quad = \quad \sum_{\mathbf{y}} \alpha_i(\mathbf{y}) \sum_s \Delta_s(\mathbf{y}_s^i, \mathbf{y}_s) \tag{53}$$

$$= \sum_s \sum_{(\mathbf{y}_1,\ldots,\mathbf{y}_S)} \alpha_i(\mathbf{y}) \Delta_s(\mathbf{y}_s^i, \mathbf{y}_s) \tag{54}$$

$$= \sum_s \sum_{\mathbf{y}_s} \Delta_s(\mathbf{y}_s^i, \mathbf{y}_s) \sum_{\sim \mathbf{y}_s} \alpha_i(\mathbf{y}) \tag{55}$$

$$= \sum_s \sum_{\mathbf{y}_s} \mu_i(\mathbf{y}_s) \Delta_s(\mathbf{y}_s^i, \mathbf{y}_s) \tag{56}$$

where we use the marginal density

$$\mu_i(\mathbf{y}_s) := \sum_{\sim \mathbf{y}_s} \alpha_i(\mathbf{y}). \tag{57}$$

The expectation of the feature function, which also appears in the objective of the dual, can be re-written analogously, using appropriate marginals defined on subsets of the components of $\mathbf{y}$, allowing us to express the objective function in terms of the marginals $\mu_i(\mathbf{y}_s)$. The trick now is to regard these marginals as new *variables*, and to re-express the dual program in terms of these. The number of these variables is only polynomial in the size of the largest clique, and so is the resulting dual. For the new program to be equivalent to the original one, the marginals have to be consistent, that is they have to be marginals of legal corresponding densities $\alpha_i(\mathbf{y})$. If the underlying graph is triangulated [16], then two conditions that ensure that this is the case are that (i) the marginals are themselves legal densities, that is they are positive and normalized:

$$\sum_{\mathbf{y}_s} \mu_i(\mathbf{y}_s) = C, \quad \mu_i(\mathbf{y}_s) \geq 0, \quad \forall i, \tag{58}$$

and (ii) the marginals agree on components they share: Consider any pair of marginals $\mu_i(\mathbf{y}_s)$, $\mu_i(\mathbf{y}_t)$ whose arguments $\mathbf{y}_s$ and $\mathbf{y}_t$ have one or more variables in common, and let us stack these common variables in a vector $\mathbf{y}_u$. For any such pair it then has to hold that:

$$\sum_{\sim \mathbf{y}_u} \mu_i(\mathbf{y}_s) = \sum_{\sim \mathbf{y}_u} \mu_i(\mathbf{y}_t). \tag{59}$$

If the underlying graph is not triangulated, then the local consistency enforced by constraints 58 and 59 is not sufficient to guarantee global consistency and it is not guaranteed that the marginals correspond to a legal density on the graph. In analogy to variational methods, we might still use the marginals, and solve a relaxed version of the problem – without guarantees, however, on the quality of the solution. To obtain solutions efficiently, we can adapt SMO (see Section 2.1.3) to the 'marginalized' program [65]. The dual representation of the weight vector after having solved the program is given by

$$\mathbf{w} = \sum_i \sum_s \sum_{\mathbf{y}_s} \mu_i(\mathbf{y}_s) \left(\phi_s(\mathbf{x}^i, \mathbf{y}_s^i) - \phi_s(\mathbf{x}^i, \mathbf{y}_s) \right). \tag{60}$$

Cutting plane approach: [68] describe an alternative approach to dealing with the exponential size quadratic program (Eq. 49). Instead of solving the program directly, we can solve a sequence of programs that contain only small subsets of the constraints of the original one. The exact algorithm works as follows: For each data-point we maintain a working set S_i of active constraints. In several sweeps over the dataset, we then determine successively for each data-point the potentially most violated constraint by finding $\hat{y} = \arg\max_y H(y)$, where

$$H(y) := \left(1 - \mathbf{w}^T \left(\phi(\mathbf{x}^i, \mathbf{y}^i) - \phi(\mathbf{x}^i, \mathbf{y})\right)\right) \Delta(\mathbf{y}^i, \mathbf{y}), \quad \text{for the margin rescaling approach, or} \tag{61}$$

$$H(y) := \Delta(\mathbf{y}^i, \mathbf{y}) - \mathbf{w}^T (\phi(\mathbf{x}^i, \mathbf{y}^i) - \phi(\mathbf{x}^i, \mathbf{y})), \quad \text{for the slack rescaling approach,} \tag{62}$$

and where $\mathbf{w}$ is given in the dual representation (Eq. 51). Note that the 'arg-max' can be obtained efficiently, if Δ and ϕ decompose as above. We also set $\xi_i = \max\{0, \max_{y \in S_i} H(\mathbf{y})\}$. Then, if the constraint violation is larger then a pre-set threshold ϵ, ie. if $H(y) > \xi_i + \epsilon$, we add the corresponding constraint to the working set for that point, and solve the quadratic program (Eq. 49 or 50) using all constraints that currently reside in the working sets. We obtain a current solution vector α with dimensionality equal to the number of constraints currently used. The procedure is repeated, until no working set S_i changes in a whole sweep over the data. [68] show that the procedure terminates after a number of iterations that is polynomial in the number of variables. The algorithm is a generalized version of a similar algorithm, that has been originally proposed for sequence alignment and is discussed in more detail in [29].

Perceptron learning: As in Section 2.1, we can use the perceptron learning rule as a simpler alternative to solving a constraint optimization problem. We can directly adapt the algorithm described in Section 2.1.3 to the multi-label case [13]. Since it requires to solve for $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$, tractability again hinges on the decomposition of $\phi(\mathbf{x}, \mathbf{y})$ wrt. $\mathbf{y}$. The perceptron learning rule for structured prediction is simpler than the corresponding constraint optimization problems. Intractabilities can be dealt with by solving standard inference problems in the model, instead of indirectly by using specialized constraint optimization procedures. Another potential advantage is the possibility of performing online learning. For the application of the perceptron learning rule in structured problems see [14] and [3].

Other methods: Efficient optimization for structured supervised models is an active area of research and several approaches in addition to the ones mentioned have been suggested, among them boosting (see [4], e.g.) and exponentiated gradient algorithms ([8]). A rigorous comparison of the different methods with respect to efficiency or accuracy does not exist and could be a subject of future research.

3.4 Kernels

The direct connection of structured models to linear classification allows us to use kernels as in Section 2.1.4. The dual representation of the compatibility measure, due to the representer theorem, takes a form similar as in Eq. 16:

$$c(\mathbf{x}, \mathbf{y}) = \sum_j \sum_{\mathbf{y}'} \alpha_j(\mathbf{y}') k((\mathbf{x}, \mathbf{y}), (\mathbf{x}^j, \mathbf{y}')). \tag{63}$$

But the sum over $\mathbf{y}'$ is in this case usually intractable. In analogy to the decomposition of the feature vectors in Section 3.1.1, however, we can obtain a tractable representation by making use of a corresponding **kernel decomposition**: Defining $k((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}'))$ as a sum over kernels that reference only cliques of $\mathbf{y}$:

$$k((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = \sum_{s,t} k_{st}((\mathbf{x}, \mathbf{y}_s), (\mathbf{x}', \mathbf{y}'_t)), \tag{64}$$

we can express Eq. 63 in terms of a polynomial number of *marginal variables* $\mu_i(\mathbf{y}_s) := \sum_{\sim \mathbf{y}_s} \alpha_i(\mathbf{y})$. The derivation is analogous to the derivation in Eqs. 53-57, but due to its fundamental importance we pursue it again:

$$\begin{aligned}
c(\mathbf{x}, \mathbf{y}) &= \sum_j \sum_{\mathbf{y}'} \alpha_j(\mathbf{y}') \sum_{s,t} k_{st}((\mathbf{x}, \mathbf{y}_s), (\mathbf{x}^j, \mathbf{y}'_s)) &(65)\\
&= \sum_j \sum_{s,t} \sum_{\mathbf{y}'_t} k_{st}((\mathbf{x}, \mathbf{y}_s), (\mathbf{x}^j, \mathbf{y}'_t)) \sum_{\sim \mathbf{y}'_t} \alpha_j(\mathbf{y}') &(66)\\
&=: \sum_j \sum_{s,t} \sum_{\mathbf{y}'_t} \mu_j(\mathbf{y}'_t) k_{st}((\mathbf{x}, \mathbf{y}_s), (\mathbf{x}^j, \mathbf{y}'_t)). &(67)
\end{aligned}$$

For the regularizer $\|\mathbf{w}\|^2$ we similarly obtain the marginalized expression:

$$\|\mathbf{w}\|^2 = \sum_{i,j} \sum_{s,t} \sum_{\mathbf{y}_s, \mathbf{y}'_t} \mu_i(\mathbf{y}_s) \mu_j(\mathbf{y}'_t) k_{st}((\mathbf{x}^i, \mathbf{y}_s), (\mathbf{x}^j, \mathbf{y}'_t)). \tag{68}$$

We can now plug these representations into any standard objective function and solve for the marginal parameters $\mu_i(\mathbf{y}_s)$, thereby fitting the model in a Reproducing Kernel Hilbert Space. Using as loss-function, for example, the log-loss, we obtain a **kernel conditional random field**, with objective function:

$$\mathcal{L}(\alpha) = \sum_{i,j} \sum_{s,t} \sum_{\mathbf{y}_s, \mathbf{y}'_t} \mu_i(\mathbf{y}_s) \mu_j(\mathbf{y}'_t) k_{s,t}((\mathbf{x}^i, \mathbf{y}_s), (\mathbf{x}^j, \mathbf{y}'_t)) + \tag{69}$$

$$\sum_i \log \sum_{\mathbf{y}} \exp \left(\sum_j \sum_{s,t} \sum_{\mathbf{y}'_t} \mu_j(\mathbf{y}'_t) k_{st}((\mathbf{x}^i, \mathbf{y}_s), (\mathbf{x}^j, \mathbf{y}'_t)) \right) - \tag{70}$$

$$\sum_j \sum_{s,t} \sum_{\mathbf{y}'_t} \mu_j(\mathbf{y}'_t) k_{st}((\mathbf{x}^i, \mathbf{y}^i_s), (\mathbf{x}^j, \mathbf{y}'_t)). \tag{71}$$

We can now perform parameter estimation by using gradient based optimization. Since the objective function, and its gradient, involve the sum over $\mathbf{y}$, however, we need to exploit the kernel decomposition a second time to be able to evaluate the functions in polynomial time. The derivation is a straightforward application of belief propagation: The kernel decomposition, similar as in an MRF, allows us to express the partition function as the sum of a product of potential functions. We can therefore use the 'sum-product' version of belief propagation to compute it. For recent work on kernel CRFs, including approaches for efficient parameter estimation, see [38], [6], [5]. The authors of [5] also point out that we can interpret the model as a kind of Gaussian process classifier (which is true for kernel logistic regression models in general).

For (differentiable) loss functions other than the log-loss we encounter analogous issues, that can be dealt with in exactly the same way. Note that for margin-based losses, because of their non-differentiability, it is more convenient to approach the parameter estimation problem via constrained optimization (see Sections 2.1.3 and 3.3.1). However, the result is similar in both cases: We obtain a representation in terms of dual variables, that we express using *marginal variables* for tractability. Note that for margin based methods the feature functions enter the objective functions and constraints only in terms of inner products. In Sections 2.1.3 and 3.3.1 we could therefore simply replace these by kernels to fit the models in kernel feature spaces. For the log-loss and other losses that lead to unconstrained optimization problems on the other hand we had to invoke the representer theorem instead.

Note that in general, when using a dual representation with decomposing kernels, we implicitly make a stationarity assumption, ie. we consider models whose parameters are tied across cliques. This is a simple result of the fact that we make use of decompositions in the original data domain, ie. *before* implicitly (and nonlinearly) mapping the data and weight vector into the feature space.

Kernel design: To apply kernels in practice we have to define joint kernel functions that decompose according to Eq. 64. The idea of using joint kernels in machine learning is extremely young, and as yet hardly any work has been reported on the design of these. The kernels that have been suggested so far are used for sequence-structured models ([5], [3], [65]) and take the form:

$$k((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = \sum_{s,t} \delta_{y_s, y'_t} k^x_s(\mathbf{x}_s, \mathbf{x}'_t) + \delta_{y_{s-1}, y'_{t-s}} \delta_{y_s, y'_t} \tag{72}$$

where $k^x_s(\mathbf{x}, \mathbf{x}')$ is some kernel function (such as an RBF-kernel) defined on the input, and $\delta_{\mathbf{y}_s, \mathbf{y}'_t}$ is used to compare the pair-wise labellings of $\mathbf{y}$. In most real-world problems that have been considered, the inputs decompose along with the outputs. (Consider for example a handwriting recognition task where several images of letters are classified jointly.) The input kernels in these problems are usually chosen to decompose, too, that is, they are functions of only partial inputs x_s. Note, however that an input-decomposition is not actually required for tractability. It could therefore be useful to consider kernels that can pick up global structure, beyond single inputs. In a handwriting recognition task those kernels could be used, for example, to model writer-dependent style variations. Future research will be necessary to evaluate the feasibility and usefulness of such an approach.

For a recent general discussion of joint kernels see [71]. Models that make use of joint kernel maps to define mappings between arbitrarily structured objects have also been used in manifold learning (see e.g. [72]). The idea of these methods is to embed input and output objects in (usually low-dimensional) Euclidean spaces and to define mappings between the objects by means of mappings between these Euclidean representations. After applying such a mapping, a *pre-image* for the embedded point in the original output-space needs to be found, which is a non-trivial task in general. For continuous output spaces the problem can usually be solved more easily than for discrete ones. Most embedding methods have therefore been defined to operate only in these. We discuss these briefly in Section 3.6.

3.5 Feature induction

The use of features $\phi(\mathbf{x}, \mathbf{y})$ to define a model is convenient, because it allows us to deal with non-vectorial data and to include arbitrary properties of input-output pairs into the compatibility measure. We can incorporate any *potentially* relevant properties into the model, and parameter estimation automatically determines the actual relevance by weighting the features accordingly. A potential problem with this approach, however, is that, since we often do not know a priori how relevant each property will be, the dimensionality of the feature vector can be exceedingly large.

An obvious way to solve this problem is to use only those features that are jointly *most relevant*. However, in practice we usually cannot assess all joint combinations of features, because of the combinatorial explosion. A heuristic to use in practice is to define a large set of potentially relevant features, and to greedily add elements from this set to the model ([7], [19]). A useful measure of the usefulness of some feature g is the increase in log-likelihood that adding it to a given model can cause. Note, that to determine this potential increase we need to compute for every candidate feature the optimal weight vector $(\mathbf{w}, w_g)$, where w_g is the weight for the candidate. A more efficient heuristic is to find only the optimal w_g, while holding $\mathbf{w}$ fixed. Denoting the log-likelihood of the model that includes the new feature by $\mathcal{L}(\mathbf{w}, w_g)$, the potential increase can then be written as

$$G(g) = \max_{w_g} \mathcal{L}(\mathbf{w}, w_g) - \mathcal{L}(\mathbf{w}) - \lambda w_g^2. \tag{73}$$

The increase in log-likelihood is usually referred to as 'gain' and it is the most common heuristic for feature induction. A reason is that for non-conditional maximum entropy models with binary features a closed form for the maximum exists [19]. For conditional models we need to resort to iterative (but only one-dimensional) optimization. To determine the optimal feature, the optimization needs to be repeated for each candidate, which can be expensive. [7] suggest some further approximations for performing it more efficiently in conditional models.

3.6 Continuous valued and energy based models

The models we have discussed so far are based on discrete outputs. In principle, it is straightforward to extend models to deal with real valued outputs: All we need to do is define corresponding features $\phi(\mathbf{x}, \mathbf{y})$ or kernels $k((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}'))$ on real vectors $\mathbf{y}$. Inference, however, in models with real valued nodes, and therefore also learning, poses quite different challenges than for discrete-valued models, and it is also less well explored. For non-Gaussian probabilistic models inference is often performed using specialized sampling methods, such as particle filters or 'nonparametric belief propagation' [62], or by using variational techniques such as expectation propagation [48].

Note that problems with real-valued output nodes could be also be treated as high-dimensional regression problems. While there are many classical approaches to solving these, such as neural networks and partial least squares regression, it is difficult to make these methods Bayesian wrt. the outputs, while at the same time treating parameter estimation in the standard, non-probabilistic, way. Bayesian models such as Gaussian processes, on the other hand, are generally defined for only one dimensional output spaces. Some recent work has tried to extend these to multi-dimensional output spaces (e.g. [66]). It points to a direction for potential further research on continuous valued models.

Several non-Bayesian, non-parametric approaches to high-dimensional regression have also been suggested from the perspective of *manifold learning* ([24], [45], [46]). As mentioned above, these methods are related to the models discussed here, since they are based on joint kernel functions. Jointly embedding the input and output in low-dimensional spaces makes it possible to capture possible correlations and that way to overcome the 'curse of dimensionality'. However, as mentioned previously, neither probabilistic modeling, nor decompositions of the output, have been considered in these kinds of models yet. A possible incorporation of these into manifold based approaches could be another subject of future research.

[41] suggest a general approach to 'end-to-end'-training of complex models, called 'energy based learning', that generalizes the probabilistic and margin-based approaches discussed in this review. They point out that loss functions other than the structured 'log'-loss or 'hinge'-loss can be brought to bear for constructing these models. Systems with continuous outputs are just special cases within this framework. An application to a face-detection problem is described in [52].

4 Applications and open research problems

In the following we discuss problems in the area of structured response models that require further investigation and therefore provide pointers to possible research topics. An important area of possible future research concerns applications. We discuss these in a separate section in the following and describe more general research problems in Section 4.2.

4.1 Applications

The applicability of SRMs is rather general and includes many problems that require complex models, and for which training data is available. Because the research area is very young, the number of existing applications is still relatively small and leaves much scope for future work, even though the number of reported applications has begun to explode in the last two years.

Until now, most applications of SRMs have been reported in natural language and some in image processing tasks. In natural language processing they have been used for, among others, shallow parsing [61], named entity recognition [43], pitch accent prediction [23]. In image processing they have been applied mainly for the tasks of object detection and recognition (see e.g. [67], [36], [25], [55]). In practically all reported applications conditional models have shown to (at times greatly) improve the state of the art in the respective areas.

Many applications of these models can be classified as methods for *machine perception*. These methods have shown in general to profit greatly from generative models, because the latter make it possible to deal with contextual information in a principled way. We believe that conditional models can be powerful competitors for constructing perceptual systems, since they allow us to specify *by example*, what we would like a percept in a particular situation to be, while at the same time being able to deal with context effects as gracefully as generative models do. While a generative face recognition system, for example, would define how the pixels of an image are determined by variables such as gender, pose and lighting effects, a conditional model could define instead how the image is to be *decomposed* into these meaningful components.

4.2 Open problems

In the following we provide some pointers to potential future research topics other than applications. See also [64] for a similar list and some additional pointers.

Optimization: Efficient parameter estimation methods are crucial for the applicability of SRMs to many real world problems. Although several optimization methods have been suggested for this purpose, further research will be necessary to develop faster algorithms and to evaluate which kind of method is most useful for which task. Another open question in this context is, how the use of variational methods in the inference sub-problems in parameter estimation interferes with parameter estimation and in what way they can affect convergence.

Hidden variables: As mentioned before, the use of latent variables in SRMs has been largely unexplored until now, even though it is promising as a means of combining discriminative learning with unsupervised structure discovery. To what degree hidden variables can be used in SRMs in order to perform a kind of task dependent pre-processing as suggested above is largely an unsolved question and requires further research.

Kernel learning and feature induction: Recently, a lot of research has been conducted on automatically learning a kernel for classification problems (see e.g. [39]). The problem of learning a kernel is in some sense 'dual' to the problem of inducing features for random fields. An adaptation of kernel learning approaches to SRMs could be useful in many applications and might help clarify this connection. An important difference for structured models, that makes such an adaptation a non-trivial task, is that for these the kernel is defined on the joint input-output space.

Semi-supervised learning: One of the main difficulties in applying an SRM in a real world task is the necessity of collecting training data. In supervised learning a lot of research has recently focused on making use of unlabeled data for training. SRMs can profit from semi-supervised approaches, because they are supervised methods themselves. The adaptation of existing approaches or the invention of new methods for semi-supervised learning of SRMs therefore provides a possible area of future research. Several recent approaches in semi-supervised learning are closely related to the problem of learning a kernel (see [73] and references therein), which points to an interesting connection between these two problems.

Structure learning: A general research problem in the area of graphical models regards the automatic inference of *model structure* as opposed to just the model parameters. Structure learning is of similar interest in SRMs and might provide an interesting way of completely automatically constructing a model from training data alone. Tree-based methods could be of particular interest for conditional models, because for fully observed trees the structure learning problem can be solved in closed form [31].

Continuous valued models: Until now, research on SRMs has focused on discrete models. In principle, most approaches discussed in this review could be directly adapted to continuous settings, as mentioned in Section 3.6. By using Gaussian potentials in a probabilistic model, for example, we could obtain a kind of 'conditional Kalman filter'; by using non-Gaussian potentials more general dynamical systems, that are trained 'by example'. Since both, continuous valued models themselves and their potential usefulness in practice, have been hardly explored yet, much further research is required in this area.

5 Conclusions

A classical way of building complex intelligent systems is by using generative models and invoking Bayes' rule for inference. The models we discussed in this review achieve a similar goal, but instead of using Bayes' rule, they directly model the response of the system to given inputs. Because the response can be structured and probabilistic, these models do not have to give up on useful properties of generative models and are, in fact, still Bayesian, if they are defined probabilistically. Since SRMs are trained supervised, they come with theoretical guarantees regarding generalization, and training is rather straightforward. In some sense, they relocate the difficulties in the design of complex system more towards the collection of training data and away from model construction itself, because they abandon the task of modeling the data.

So far, most applications have focused on sequence structured models, that are used to solve problems for which HMMs have been used traditionally. Only a few models have been suggested yet that extend the applicability to more general problems, such as those that arise in image processing tasks and computer vision. We believe that the applicability of these models can be further extended and be brought to use in additional problems in artificial intelligence.

As this review has tried to make clear, there is really only one structured response model, and it is based on the definition of a compatibility measure that decomposes according to a graph defined on the system variables. We have shown that several variations of the basic idea are possible, leading to certain desirable properties such as probabilistic semantics or sparsity. We do not believe that any particular of these variations will generally turn out to be superior over any other in practice, but rather that enhancements of this modeling framework in general, such as by the use of novel optimization- and feature induction approaches, or the incorporation of new approaches for semi-supervised learning, will be most useful for future applications.

6 Acknoledgements

I thank the members of my advisory committee, Geoffrey Hinton, Radford Neal, Sam Roweis and Rich Zemel for helpful comments on an earlier version of this report.

References

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. pages 635–649, 1988.

[2] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[3] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines, 2003.

[4] Yasemin Altun, Thomas Hofmann, and Mark Johnson. Discriminative learning for label sequences via boosting. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press.

[5] Yasemin Altun, Thomas Hofmann, and Alexander J. Smola. Gaussian process classification for segmenting and annotating sequences. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 4, New York, NY, USA, 2004. ACM Press.

[6] Yasemin Altun, Alex J. Smola, and Thomas Hofmann. Exponential families for conditional random fields. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 2–9, Arlington, Virginia, United States, 2004. AUAI Press.

[7] McCallum Andrew. Efficiently inducing features of conditional random fields. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 403–410, San Francisco, CA, 2003. Morgan Kaufmann Publishers.

[8] Peter L. Bartlett, Michael Collins, Ben Taskar, and David McAllester. Exponentiated gradient algorithms for large-margin structured classification. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 113–120. MIT Press, Cambridge, MA, 2005.

[9] Yoshua Bengio and Paolo Frasconi. An input output HMM architecture. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 427–434. The MIT Press, 1995.

[10] A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. A maximum entropy approach to natural language processing. Technical report, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY, 10598, 1994.

[11] Léon Bottou. *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis, Université de Paris XI, Orsay, France, 1991.

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, New York, 2004.

[13] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms, 2002.

[14] Michael Collins. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *IWPT*. Tsinghua University Press, 2001.

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[16] Robert G. Cowell, Steffen L. Lauritzen, A. Philip David, David J. Spiegelhalter, and David J. Spiegelhater. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[17] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, 2002.

[18] Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.

[19] Stephen Della Pietra, Vincent J. Della Pietra, and John D. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[20] Thomas G. Dietterich. Machine learning for sequential data: A review. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30, London, UK, 2002. Springer-Verlag.

[21] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Computational Learing Theory*, pages 209–217, 1998.

[22] Brendan J. Frey and Nebosja Jojic. A Comparison of Algorithms for Inference and Learning in Probabilistic Graphical Models. Available at http://www.psi.toronto.edu/ frey/stuff/tutorial.ps.gz.

[23] Michelle L. Gregory and Yasemin Altun. Using conditional random fields to predict pitch accents in conversational speech. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, 2004.

[24] J. H. Ham, D. D. Lee, and L. K. Saul. Learning high dimensional correspondences from low dimensional manifolds. In *In Proceedings of the ICML 2003 Workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pages 34–41, Washington, D.C., 2003.

[25] Xuming He, Richard S. Zemel, and Miguel Á. Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *CVPR (2)*, pages 695–702, 2004.

[26] G. E. Hinton and T. J. Sejnowski. Learning and relearning in boltzmann machines. pages 282–317, 1986.

[27] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, 2002.

[28] Zhu Ji and Hastie Trevor. Kernel logistic regression and the import vector machine. *Journal of Computational & Graphical Statistics*, 14(1):185–205, March 2004.

[29] Thorsten Joachims. Learning to align sequences: A maximum-margin approach. Technical report, Cornell University, 2003.

[30] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

[31] Michael Jordan. *An Introduction to Probabilistic Graphical Models*.

[32] S. Kakade, Y.W. Teh, and S. Roweis. An alternate objective function for Markovian fields. In *Proceedings of the International Conference on Machine Learning*, volume 19, 2002.

[33] G. S. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *J. Math. Anal. Applications*, 33(1):82–95, 1971.

[34] Kschischang, Frey, and Loeliger. Factor graphs and the sum-product algorithm. *IEEETIT: IEEE Transactions on Information Theory*, 47, 2001.

[35] F. R. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2):498–519, 2001.

[36] Sanjiv Kumar and Martial Hebert. Discriminative random fields: A discriminative framework for contextual interaction in classification. In *Proceedings of the 2003 IEEE International Conference on Computer Vision (ICCV '03)*, volume 2, pages 1150–1157, 2003.

[37] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[38] John Lafferty, Xiaojin Zhu, and Yan Liu. Kernel conditional random fields: representation and clique selection. In *ICML '04: Twenty-first international conference on Machine learning*. ACM Press, 2004.

[39] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *J. Mach. Learn. Res.*, 5:27–72, 2004.

[40] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, November 1998.

[41] Yann LeCun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In *Proc. of the 10-th International Workshop on Artificial Intelligence and Statistics (AIStats'05)*, 2005.

[42] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of CoNLL-2002*, pages 49–55. Taipei, Taiwan, 2002.

[43] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, 2003.

[44] Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proc. 17th International Conf. on Machine Learning*, pages 591–598. Morgan Kaufmann, San Francisco, CA, 2000.

[45] R. Memisevic. Kernel information embeddings, 2005.

[46] Roland Memisevic and Geoffrey Hinton. Multiple relational embedding. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 913–920. MIT Press, Cambridge, MA, 2005.

[47] Thomas P. Minka. Algorithms for maximum-likelihood logistic regression. Technical report, CMU Statistics Department, 2001.

[48] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[49] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[50] Radford M. Neal and Geoffrey E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. MIT Press, Cambridge, MA, USA, 1999.

[51] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, August 1999.

[52] Margarita Osadchy, Matthew L. Miller, and Yann LeCun. Synergistic face detection and pose estimation with energy-based models. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1017–1024. MIT Press, Cambridge, MA, 2005.

[53] John C. Platt. Using analytic qp and sparseness to speed training of support vector machines. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 557–563, Cambridge, MA, USA, 1999. MIT Press.

[54] Yuan Qi, Martin Szummer, and Tom Minka. Bayesian conditional random fields. In Robert G. Cowell and Zoubin Ghahramani, editors, *aistats05*, pages 269–276. Society for Artificial Intelligence and Statistics, 2005.

[55] Ariadna Quattoni, Michael Collins, and Trevor Darrel. Conditional random fields for object recognition. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005.

[56] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, pages 267–296. Kaufmann, San Mateo, CA, 1990.

[57] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. pages 267–296, 1990.

[58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[59] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005.

[60] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[61] F. Sha and F. Pereira. Shallow parsing with conditional random fields, 2003.

[62] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation, 2002.

[63] Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: factorized probabilistic models for labeling and segmenting sequence data. In *ICML '04: Twenty-first international conference on Machine learning*. ACM Press, 2004.

[64] Ben Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, CA, 2004.

[65] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[66] Y.W. Teh, M. Seeger, and M.I. Jordan. Semiparametric latent factor models. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, volume 10, 2005.

[67] Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Contextual models for object detection using boosted random fields. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005.

[68] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML '04: Twenty-first international conference on Machine learning*, New York, NY, USA, 2004. ACM Press.

[69] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, N.Y., 1995.

[70] Hanna Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

[71] J. Weston, B. Schölkopf, and O. Bousquet. Joint kernel maps. volume LNCS 3512, pages 176–191, Berlin Heidelberg, Germany, 2005. Springer-Verlag.

[72] Jason Weston, Olivier Chapelle, Andre Elisseeff, Bernhard Schölkopf, and Vladimir Vapnik. Kernel dependency estimation. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 873–880. MIT Press, Cambridge, MA, 2003.

[73] Xiaojin Zhu, Jaz Kandola, Zoubin Ghahramani, and John Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1641–1648. MIT Press, Cambridge, MA, 2005.