Machine Learning for Sequential Data: A Review

Thomas G. Dietterich

Oregon State University, Corvallis, Oregon, USA,
tgd@cs.orst.edu,
WWW home page: http://www.cs.orst.edu/~tgd

Abstract. Statistical learning problems in many fields involve sequential data. This paper formalizes the principal learning tasks and describes the methods that have been developed within the machine learning research community for addressing these problems. These methods include sliding window methods, recurrent sliding windows, hidden Markov models, conditional random fields, and graph transformer networks. The paper also discusses some open research issues.

1 Introduction

The classical supervised learning problem is to construct a classifier that can correctly predict the classes of new objects given training examples of old objects [19]. A typical application is in optical character recognition where the objects are images of hand-written characters and the classes are the 26 alphabetic letters. The classifier takes an image as input and produces a letter as output. This task is typically formalized as follows.

Let $\mathbf{x}$ denote an image of a hand-written character and $y \in \{A, \ldots, Z\}$ denote the corresponding letter class. A training example is a pair $(\mathbf{x}, y)$ consisting of an image and its associated class label. We assume that the training examples are drawn independently and identically from the joint distribution $P(\mathbf{x}, y)$, and we will refer to a set of N such examples as the training data.

A classifier is a function h that maps from images to classes. The goal of the learning process is to find an h that correctly predicts the class $y = h(\mathbf{x})$ of new images $\mathbf{x}$. This is accomplished by searching some space $\mathcal{H}$ of possible classifiers for a classifier that gives good results on the training data without overfitting.

Over the past 10 years, supervised learning has become a standard tool in many fields, and practitioners have learned how to take new application problems and view them as supervised learning problems. For example, in cellular telephone fraud detection, each $\mathbf{x}$ describes a telephone call, and y is 0 if the call is legitimate and 1 if the call originated from a stolen (or cloned) cell phone [8]. Another example involves computer intrusion detection where each x describes a request for a computer network connection and y indicates whether that request is part of an intrusion attempt. A third example is part-of-speech tagging in which each $\mathbf{x}$ describes a word and each y gives the part-of-speech of that word (noun, verb, adjective, etc.).

One thing that is apparent in these (and other) applications is that they do not quite fit the supervised learning framework. Rather than being drawn independently and identically (iid) from some joint distribution $P(\mathbf{x}, y)$, the training data actually consist of *sequences* of $(\mathbf{x}, y)$ pairs. These sequences exhibit significant sequential correlation. That is, nearby $\mathbf{x}$ and y values are likely to be related to each other.

For example, before a cell phone is stolen, all of the y values will be 0. Afterwards, all of the y values will be 1. Similarly, computer intrusions exhibit significant clustering—particularly denial of service attacks. Other kinds of attacks are deliberately spaced over time to avoid detection, which is a form of temporal anti-correlation. In part-of-speech tagging, sequences of parts of speech are constrained by the grammar of the language. Hence, in English, a sequence such as (verb verb adjective verb verb) would be very unlikely. Sequential patterns are present even in the original task of character recognition: Character sequences usually form words rather than random sequences of letters.

Sequential patterns are important because they can be exploited to improve the prediction accuracy of our classifiers. In English, for example, if the classifier determines that one letter is Q, then the next letter is almost certain to be U. In telephone fraud detection, it is only possible to detect fraud by looking at the distribution of typical (legitimate) phone calls and then to see that this distribution changes when the telephone is stolen. Any single phone call, viewed in isolation, appears to be perfectly legitimate.

The *sequential supervised learning* problem can be formulated as follows. Let $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ be a set of N training examples. Each example is a pair of sequences $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i = \langle \mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{i,T_i} \rangle$ and $\mathbf{y}_i = \langle y_{i,1}, y_{i,2}, \ldots, y_{i,T_i} \rangle$. For example, in part-of-speech tagging, one $(\mathbf{x}_i, \mathbf{y}_i)$ pair might consist of $\mathbf{x}_i = \langle$do you want fries with that$\rangle$ and $\mathbf{y}_i = \langle$verb pronoun verb noun prep pronoun$\rangle$. The goal is to construct a classifier h that can correctly predict a new label sequence $\mathbf{y} = h(\mathbf{x})$ given an input sequence $\mathbf{x}$.

This task should be contrasted with two other, closely-related tasks. The first of these is the time-series prediction problem. Here the task is to predict the $t+1^{\text{st}}$ element of a sequence $\langle y_1, \ldots, y_t \rangle$. This can be extended in two ways. First, we can consider the case where each y_t is a vector $\mathbf{y}_t$. The time-series task becomes to predict simultaneously a whole collection of parallel time series: Predict $\mathbf{y}_{t+1}$ given $\langle \mathbf{y}_1, \ldots, \mathbf{y}_t \rangle$. Second, we can consider the case when there are other "features" or co-variates $\langle \mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{x}_{t+1} \rangle$ available.

There are two key differences between time-series prediction and sequential supervised learning. First in sequential supervised learning, the entire sequence $\langle \mathbf{x}_1, \ldots, \mathbf{x}_T \rangle$ is available before we make any predictions of the y values, whereas in time-series prediction, we have only a prefix of the sequence up to the current time $t+1$. Second, in time-series analysis, we have the true observed y values up to time t, whereas in sequential supervised learning, we are not given any y values and we must predict them all.

The second closely-related task is sequence classification. In this task, the problem is to predict a single label y that applies to an entire input sequence

$\langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T \rangle$. For example, given a sequence of images of hand-written characters, the task might be to determine the identity of the person who wrote those characters (hand-writing identification). In these kinds of problems, each training example consists of a pair $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is a sequence $\langle \mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,T_i} \rangle$ and each y_i is a class label (such as a person's identification number). A similar problem arises in recognizing whole words on handwritten checks. The $\mathbf{x}_i$ could be a sequence of hand-written letters, and y_i could be a word such as "hundred".

All of these problems are closely related, and sometimes a solution to one can be converted into a solution for another. For example, one strategy for recognizing a handwritten word (e.g., "hundred") would be first to solve the sequential supervised learning problem of recognizing the individual letters ⟨H, U, N, D, R, E, D⟩, and then assembling them into the entire word. This works for cases where the class label y can be decomposed into sub-parts (in this case, individual letters). But no similar strategy would work for recognizing an individual's identity from their handwriting.

Similarly, some methods for sequential supervised learning make their predictions by scanning the sequence from left-to-right, and such methods can typically be applied to time-series problems as well. However, methods that analyze the entire sequence of $\mathbf{x}_t$ values before predicting the y_t labels typically can give better performance on the sequential supervised learning problem.

2 Research Issues in Sequential Supervised Learning

Now let us consider three fundamental issues in sequential supervised learning: (a) loss functions, (b) feature selection, and (c) computational efficiency.

2.1 Loss Functions

In classical supervised learning, the usual measure of success is the proportion of (new) test data points correctly classified. This is known as the 0/1 loss, since a loss of 1 is received for every misclassified test point and a loss of 0 for every correctly-classified test point. More recently, researchers have been studying non-uniform loss functions. These are usually represented by a cost matrix $C(i, j)$, which gives the cost of assigning label i to an example whose true label is j. In such cases, the goal is to find a classifier with minimum expected cost. One strategy for developing such a classifier is to learn a conditional density estimator $P(y|\mathbf{x})$ and then classify a new data point $\mathbf{x}$ according to the formula

$$y = \operatorname*{argmin}_i \sum_j P(j|\mathbf{x}) C(i, j).$$

This formula chooses the class whose expected cost is minimum.

In sequential supervised learning problems, many different kinds of loss functions are encountered. Statistical learning methods are needed that can minimize the expected loss for all of these different loss functions. First, we will consider some of the loss functions that have appeared in various applications. Second, we will discuss how these different loss functions might be incorporated into learning and prediction.

In some problems, the goal is to predict the entire output sequence of labels $\mathbf{y}_i$ correctly, and any error in this sequence counts as an error for the entire sequence. Other problems exhibit the opposite extreme: the goal is to predict correctly as many individual labels $y_{i,t}$ in the sequence as possible. One can imagine problems intermediate between these extremes.

In many applications, different kinds of errors have different costs. Consider cellular telephone fraud. The real goal here is to determine the time t^* at which the telephone was stolen (or cloned). As described above, we can view this as a sequential supervised learning problem in which $y_t = 0$ for $t < t^*$ and $y_t = 1$ for $t \geq t^*$. Consider the problem of making a prediction $\overline{t}$ for the value of t^*. One strategy would be to apply the learned classifier h to classify each element $\mathbf{x}_{i,t}$ and predict $\overline{t} = t$ for the earliest time t for which $h(\mathbf{x}_{i,t}) = 1$. A typical form for the loss function assesses a penalty of $c_1(t^* - \overline{t})$ if $\overline{t} < t^*$ and a penalty of $c_2(\overline{t} - t^*)$ if $\overline{t} > t^*$. In the telephone fraud case, the first penalty is the cost of lost business if we prematurely declare the telephone to be stolen. The second penalty is the cost of the fraudulent calls when we are late in declaring the telephone to be stolen. More complex loss functions can be imagined that take into account the cost of each individual telephone call. This argument applies to any form of monitoring of financial transactions. It also applies to systems that must determine when manufacturing equipment begins to malfunction.

Another kind of loss function applies to problems of event detection. Suppose that the input sequence $\mathbf{x}_i$ consists of infrequent events superimposed on "normal" signals. For example, in high-energy physics, these might be detections of rare particles. In astronomy, these might be sightings of events of interest (e.g., gamma ray bursts). The loss function should assign a cost to missed events, to extra events, and to events that are detected but not at the correct time.

Finally, a loss function closely related to event detection arises in the problem of hyphenation. Consider the problem of learning to hyphenate words so that a word processor can determine where to break words during typesetting (e.g., "porcupine" $\rightarrow$ "00101000"). In this case, the input sequence $\mathbf{x}_i$ is a string of letters, and the output sequence $\mathbf{y}_i$ is a sequence of 0's and 1's, such that $y_{i,t} = 1$ indicates that a hyphen can legally follow the letter $x_{i,t}$. Each opportunity for a hyphen can be viewed as an event. False positive hyphens are very expensive, because they lead to incorrectly-hyphenated words that distract the reader. False negative hyphens are less of a problem—provided that at least one hyphen is correctly identified. Furthermore, hyphens near the middle of long words are more helpful to the typesetting program than hyphens near the ends of the words. This is a case where the loss function involves a global analysis of the predicted sequence $\mathbf{y}_i$ but where not all of the individual y_t predictions need to be correct.

How can these kinds of loss functions be incorporated into sequential supervised learning? One approach is to view the learning problem as the task of predicting the (conditional) joint distribution of all of the labels in the output sequence: $P(\mathbf{y}_i|\mathbf{x}_i)$. If this joint distribution can be accurately predicted, then all of the various loss functions can be evaluated, and the optimal decisions can be chosen. There are two difficulties with this: First, predicting the entire joint distribution is typically very difficult. Second, computing the optimal decisions given the joint distribution may also be computationally infeasible.

Some loss functions only require particular marginal probabilities. For example, if the loss function is only concerned with the number of correct individual predictions $y_{i,t}$, then the goal of learning should be to predict the individual marginal probabilities $P(y_{i,t}|\mathbf{x}_i)$ correctly. If the loss function is only concerned with classifying the entire sequence correctly, then the goal should be to predict $\text{argmax}_{\mathbf{y}_i} P(\mathbf{y}_i|\mathbf{x}_i)$ correctly. We will see below that there are learning algorithms that directly optimize these quantities.

2.2 Feature Selection and Long-Distance Interactions

Any method for sequential supervised learning must employ some form of divide-and-conquer to break the overall problem of predicting $\mathbf{y}_i$ given $\mathbf{x}_i$ into subproblems of predicting individual output labels $y_{i,t}$ given some subset of information from $\mathbf{x}_i$ (and perhaps other predicted values $y_{i,u}$). One of the central problems of sequential supervised learning is to identify the relevant information subset for making accurate predictions.

In standard supervised learning, this is known as the *feature selection problem*, and there are four primary strategies for solving it. The first strategy, known as the wrapper approach [12], is to generate various subsets of features and evaluate them by running the learning algorithm and measuring the accuracy of the resulting classifier (e.g., via cross-validation or by applying the Akaiki Information Criterion). The feature subsets are typically generated by forward selection (starting with single features and progressively adding one feature at a time) or backward elimination (starting with all of the features and progressively removing one feature at a time). For some learning algorithms, such as linear regression, this can be implemented very efficiently.

The second strategy is to include all possible features in the model, but to place a penalty on the values of parameters in the fitted model. This causes the parameters associated with useless features to become very small (perhaps even zero). Examples of this approach include ridge regression [10], neural network weight elimination [24], and L_1-norm support vector machines (SVMs; [5]).

The third strategy is to compute some measure of feature relevance and remove low-scoring features. One of the simplest measures is the mutual information between a feature and the class. This (or similar measures) forms the basis of recursive-partioning algorithms for growing classification and regression trees. These methods incorporate the choice of relevant features into the tree-growing process [3, 21]. Unfortunately, this measure does not capture interactions between features. Several methods have been developed that identify such interactions including RELIEFF [14], Markov blankets [13], and feature racing [17].

The fourth strategy is to first fit a simple model and then analyze the fitted model to identify the relevant features. For example, Chow and Liu [4] describe an efficient algorithm for fitting a tree-structured Bayesian network to a data set. This network can then be analyzed to remove features that have low influence on the class. Kristin Bennett (personal communication, 2001) fits L_1-norm SVMs to drug binding data to remove irrelevant features prior to fitting a more complex SVM regression model.

In sequential supervised learning, most authors have assumed that a fixed-sized neighborhood of features is relevant for predicting each output value. For example, suppose we assume a neighborhood of size 3. Then we will employ $\mathbf{x}_{i,t-1}, \mathbf{x}_{i,t}$, and $\mathbf{x}_{i,t+1}$ to predict $y_{i,t}$. However, this has two drawbacks. First, not all of the features in each feature vector $\{\mathbf{x}_{i,u}\}_{u=t-1}^{t+1}$ are necessarily relevant. Second, there may be longer-range interactions that are missed. For example, consider the problem of predicting the pronunciation of English words from their spelling. The only difference between the words "thought" and "though" is the final "t", yet this influences the pronunciation of the initial "th" (changing it from unvoiced to voiced). An even more extreme case is the pair "photograph" and "photography" in which the final "y" changes the pronunciation of every vowel in the word.

Of the four feature-selection strategies discussed above, it is unlikely that the first two are feasible for sequential supervised learning. There are so many potential features to consider in a long sequence, that a direct search of possible feature subsets becomes completely intractable (even with greedy algorithms). The third and fourth approaches are more promising, but with long sequences, they still raise the possibility of overfitting. Hence, any successful methodology for feature selection (and for handling long distance interactions) will probably need to combine human expertise with statistical techniques rather than applying statistical techniques alone.

2.3 Computational Efficiency

A third challenge for sequential supervised learning is to develop methods for learning and classification that are computationally efficient. We will see below that some of the learning algorithms that have been proposed for sequential supervised learning are computationally expensive.

Even after learning, it may be computationally expensive to apply a learned classifier to make minimum-cost predictions. Even relatively efficient methods such as the Viterbi algorithm can be slow for complex models.

These computational challenges are probably easier to solve than the statistical ones. As in many other computational problems, it is usually possible to identify a series of approximate methods that are progressively more expensive and more accurate. The cheapest methods can be applied first to generate a set of possible candidate solutions which can then be evaluated more carefully by the more expensive methods.

3 Machine Learning Methods for Sequential Supervised Learning

In this section, we will briefly describe six methods that have been applied to solve sequential supervised learning problems: (a) sliding-window methods, (b) recurrent sliding windows, (c) hidden Markov models, (d) maximum entropy Markov models, (e) input-output Markov models, (f) conditional random fields, and (g) graph transformer networks.

3.1 The Sliding Window Method

The sliding window method converts the sequential supervised learning problem into the classical supervised learning problem. It constructs a *window classifier* h_w that maps an input window of width w into an individual output value y. Specifically, let $d = (w-1)/2$ be the "half-width" of the window. Then h_w predicts $y_{i,t}$ using the window $\langle x_{i,t-d}, x_{i,t-d+1}, \ldots, x_{i,t}, \ldots, x_{i,t+d-1}, x_{i,t+d}\rangle$. In effect, the input sequence $\mathbf{x}_i$ is padded on each end by d "null" values and then converted into N_i separate examples.

The window classifier h_w is trained by converting each sequential training example $(\mathbf{x}_i, \mathbf{y}_i)$ into windows and then applying a standard supervised learning algorithm. A new sequence $\mathbf{x}$ is classified by converting it to windows, applying h_w to predict each y_t and then concatenating the y_t's to form the predicted sequence $\mathbf{y}$.

The obvious advantage of this sliding window method is that permits any classical supervised learning algorithm to be applied. Sejnowski and Rosenberg [23] applied the backpropagation neural network algorithm with a 7-letter sliding window to the task of pronouncing English words. A similar approach (but with a 15-letter window) was employed by Qian and Sejnowski [20] to predict protein secondary structure from the protein's sequence of amino acid residues. Provost and Fawcett [8] addressed the problem of cellular telephone cloning by applying the RL rule learning system to day-long windows from telephone calling logs.

Although the sliding window method gives adequate performance in many applications, it does not take advantage of correlations between nearby y_t values. To be more precise, the only relationships between nearby y_t values that are captured are those that are predictable from nearby x_t values. If there are correlations among the y_t values that are independent of the x_t values, then these are not captured.

3.2 Recurrent Sliding Windows

One way that sliding window methods can be improved is to make them recurrent. In a recurrent sliding window method, the predicted value $\overline{y}_{i,t}$ is fed as an input to help make the prediction for $y_{i,t+1}$. Specifically, with a window of half-width d, the most recent d predictions, $\overline{y}_{i,t-d}, \overline{y}_{i,t-d+1}, \ldots, \overline{y}_{i,t-1}$, are used as inputs (along with the sliding window $\langle x_{i,t-d}$, $x_{i,t-d+1}$, $\ldots$, $x_{i,t}$, $\ldots$, $x_{i,t+d-1}$, $x_{i,t+d}\rangle$) to predict $y_{i,t}$.

Table 1. left-to-right and right-to-left.

Method	Direction of letter processing	% correct Level of Aggregation Word	Letter
Sliding Window		12.5	69.6
Recurrent Sliding Window	Left-to-Right	17.0	67.9
Recurrent Sliding Window	Right-to-Left	24.4	74.2

Bakiri and Dietterich [1] applied this technique to the English pronunciation problem using a 7-letter window and a decision-tree algorithm. Table 1 summarizes the results they obtained when training on 1000 words and evaluating the performance on a separate 1000-word test data set. The baseline sliding window method correctly pronounces 12.5% of the words and 69.6% of the individual letters in the words. A recurrent sliding window moving left-to-right improves the word-level performance but worsens the pronunciations of individual letters. However, a right-to-left sliding window improves both the word-level and letter-level performance. Indeed, the percentage of correct word-level pronunciations has nearly doubled!

Clearly, the recurrent method captures predictive information that was not being captured by the simple 7-letter sliding window. But why is the right-to-left scan superior? It appears that in English, the right-to-left scan is able to capture long-distance effects such as those mentioned above for "thought" and "photography". For example, the right-most window can correctly pronounce the "y" of "photography". This information is then available when the system attempts to pronounce the "a". And this information in turn is available when the system is pronouncing the second "o", and so on. Because the stress patterns in English are determined by the number of syllables to the right of the current syllable, a right-to-left recurrent window is able to correctly predict these stresses, and hence, choose the correct pronunciations for the vowels in each syllable.

One issue arises during training: What values should be used for the $\overline{y}_{i,t}$ inputs when training the window classifier? One approach would be to first train a non-recurrent classifier, and then use its $\overline{y}_{i,t}$ predictions as the inputs. This process can be iterated, so that the predicted outputs from each iteration are employed as inputs in the next iteration. Another approach is to use the correct labels $y_{i,t}$ as the inputs. The advantage of using the correct labels is that training can be performed with the standard supervised learning algorithms, since each training example can be constructed independently. This was the choice made by Bakiri and Dietterich.

In addition to recurrent decision trees, many other classifiers can be made recurrent. Recurrent neural networks are of particular interest. Figure 1 shows two of the many architectures that have been explored. Part (a) shows a network in which the output units are fed as inputs to the hidden units at the next time step. This is essentially identical to the recurrent decision trees employed by Bakiri and Dietterich, except that during training, the predicted outputs $\overline{y}_{i,t-1}$ are used as the inputs at time t. Networks similar to this were first introduced

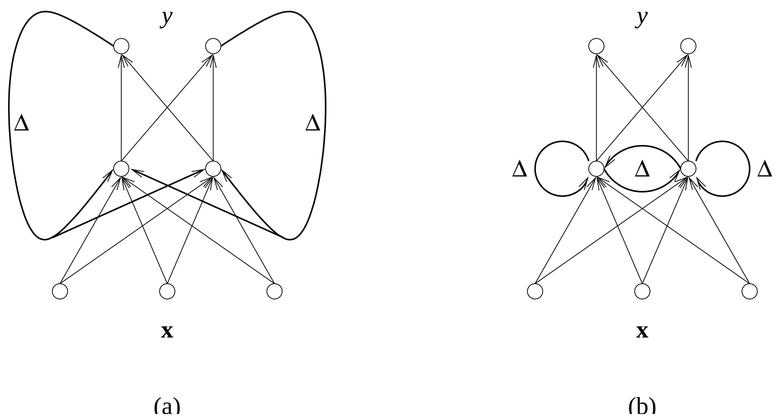

Fig. 1. Two recurrent network architectures: (a) outputs are fed back to hidden units; (b) hidden units are fed back to hidden units. The **Δ** symbol indicates a delay of one time step.

by Jordan [11]. Part (b) shows a network in which the hidden unit activations at time $t-1$ are fed as additional inputs at time t. This allows the network to develop a representation for the recurrent information that is separate from the representation of the output y values. This architecture was introduced by Elman [7].

These networks are usually trained iteratively via a procedure known as backpropagation-through-time (BPTT) in which the network structure is "unrolled" for the length of the input and output sequences $\mathbf{x}_i$ and $\mathbf{y}_i$ [22]. Recurrent networks have been applied to a variety of sequence-learning problems [9].

3.3 Hidden Markov Models and Related Methods

The hidden Markov Model (HMM; see Figure 2(a)) is a probabilistic model of the way in which the $\mathbf{x}_i$ and $\mathbf{y}_i$ strings are generated—that is, it is a representation of the joint distribution $P(\mathbf{x}, \mathbf{y})$. It is defined by two probability distributions: the transition distribution $P(y_t|y_{t-1})$, which tells how adjacent y values are related, and the observation distribution $P(\mathbf{x}|y)$, which tells how the observed $\mathbf{x}$ values are related to the hidden y values. These distributions are assumed to be stationary (i.e., the same for all times t).

In most problems, $\mathbf{x}$ is a vector of features $(x_1, \ldots, x_n)$, which makes the observation distribution difficult to handle without further assumptions. A common assumption is that each feature is generated independently (conditioned on y). This means that $P(\mathbf{x}|y)$ can be replaced by the product of n separate distributions $P(x_j|y)$, $j = 1, \ldots, n$.

The HMM generates $\mathbf{x}_i$ and $\mathbf{y}_i$ as follows. Suppose there are K possible labels $1, \ldots, K$. Augment this set of labels with a start label 0 and a terminal label $K+1$. Let $y_{i,0} = 0$. Then, generate the sequence of y values according to $P(y_{i,t}|y_{i,t-1})$ until $y_{i,t} = K+1$. At this point, set $T_i := t$. Finally, for each $t = 1, \ldots, T_i$, generate $\mathbf{x}_{i,t}$ according to the observation probabilities $P(\mathbf{x}_{i,t}|y_{i,t})$.

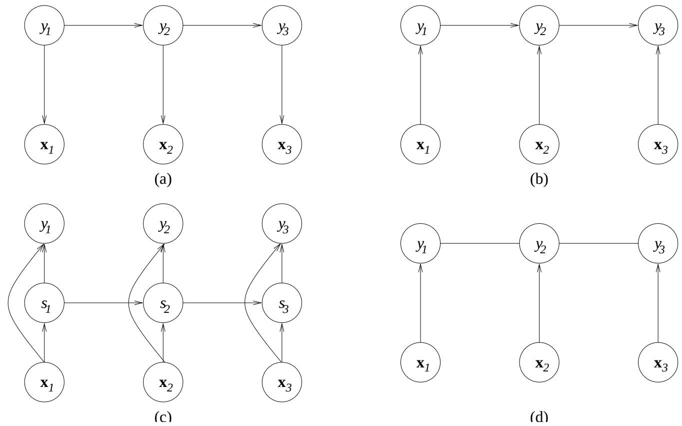

Fig. 2. Probabilistic models related to hidden Markov models: (a) HMM, (b) maximum entropy Markov model, (c) input-output HMM, and (d) conditional random field

In a sequential supervised learning problem, it is straightforward to determine the transition and observation distributions. $P(y_{i,t}|y_{i,t-1})$ can be computed by looking at all pairs of adjacent y labels (after prepending 0 at the start and appending $K+1$ to the end of each $\mathbf{y}_i$). Similarly, $P(x_j|y)$ can be computed by looking at all pairs of x_j and y.

The most complex computation is to predict a value $\overline{\mathbf{y}}$ given an observed sequence $\mathbf{x}$. This computation depends on the nature of the loss function. Because the HMM is a representation of the joint probability distribution $P(\mathbf{x}, \mathbf{y})$, it can be applied to compute the probability of any particular $\mathbf{y}$ given any particular $\mathbf{x}$: $P(\mathbf{y}|\mathbf{x})$. Hence, for an arbitrary loss function $L(\overline{\mathbf{y}}, \mathbf{y})$, the optimal prediction is

$$\overline{\mathbf{y}} = \operatorname*{argmin}_{\mathbf{z}} \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) L(\mathbf{z}, \mathbf{y}).$$

However, if the sequences are of length L and there are K labels, then direct evaluation of this equation requires $O(K^L)$ probability evaluations, which is usually impractical.

There are two notable cases where this computation can be performed in $O(K^2 L)$ time. The first is where the loss function depends on the entire sequence. In this case, the goal is usually to find the $\mathbf{y}$ with the highest probability: $\overline{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$. This can be computed via the Viterbi algorithm, which is a dynamic programming algorithm that computes, for each class label u and each time step t, the probability of the most likely path starting at time 0 end ending at time t with class u. When the algorithm reaches the end of the sequence, it has computed the most likely path from time 0 to time T_i and its probability.

The second interesting case is where the loss function decomposes into separate decisions for each y_t. In this case, the so-called Forward-Backward algorithm can be applied. It performs a left-to-right pass, which fills a table of $\alpha_t(y_t)$ values which represent $P(y_1, \ldots, y_t|\mathbf{x}_1, \ldots, \mathbf{x}_t)$, and a right-to-left pass, which fills a table of $\beta_t(y_t)$ values which represent $P(y_t, \ldots, y_{T_i}|\mathbf{x}_{t+1}, \ldots, \mathbf{x}_{T_i})$. Once these two passes are complete, the quantity

$$\gamma_t(u) = \frac{\alpha_t(u) \cdot \beta_t(u)}{\sum_v \alpha_t(v) \cdot \beta_t(v)}$$

gives the desired probability: $P(y_t = u|\mathbf{x})$. This probability can be applied to choose the predicted value $\overline{y}_t$ that minimizes the loss function.

Although HMMs provide an elegant and sound methodology, they suffer from one principal drawback: The structure of the HMM is often a poor model of the true process producing the data. Part of the problem stems from the Markov property. Any relationship between two separated y values (e.g., y_1 and y_4) must be communicated via the intervening y's. A first-order Markov model (i.e., where $P(y_t)$ only depends on y_{t-1}) cannot in general capture these kinds of relationships.

Sliding window methods avoid this difficulty by using a window of $\mathbf{x}_t$ values to predict a single y_t. However, the second problem with the HMM model is that it generates each $\mathbf{x}_t$ only from the corresponding y_t. This makes it difficult to use an input window. In theory, one could replace the output distribution $P(\mathbf{x}_t|y_t)$ by a more complex distribution $P(\mathbf{x}_t|y_{t-1}, y_t, y_{t+1})$ which would then allow an observed value $\mathbf{x}_t$ to influence the three y values. But it is not clear how to represent such a complex distribution compactly.

Several directions have been explored to try to overcome the limitations of the HMM: Maximum Entropy Markov models (MEMMs), Input-Output HMMs (IOHMMs), and conditional random fields (CRFs); see Figure 2. All of these are *conditional* models that represent $P(\mathbf{y}|\mathbf{x})$ rather than $P(\mathbf{x}, \mathbf{y})$. They do not try to explain how the $\mathbf{x}$'s are generated. Instead, they just try to predict the $\mathbf{y}$ values given the $\mathbf{x}$'s. This permits them to use arbitrary features of the $\mathbf{x}$'s including global features, features describing non-local interactions, and sliding windows.

The Maximum Entropy Markov Model learns $P(y_t|y_{t-1}, \mathbf{x}_t)$. It is trained via a maximum entropy method that attempts to maximize the conditional likelihood of the data: $\prod_{i=1}^{N} P(\mathbf{y}_i|\mathbf{x}_i)$. The maximum entropy approach represents $P(y_t|y_{t-1}, \mathbf{x}_t)$ as a log-linear model:

$$P(y_t|y_{t-1}, \mathbf{x}) = \frac{1}{Z(y_{t-1}, \mathbf{x})} \exp\left(\sum_\alpha \lambda_\alpha f_\alpha(\mathbf{x}, y_t)\right),$$

where $Z(y_{t-1}, \mathbf{x})$ is a normalizing factor to ensure that the probabilities sum to 1. Each f_α is a boolean feature that can depend on y_t and on *any* properties of the input sequence $\mathbf{x}$. For example, in their experiments with MEMMs, McCallum, et al. [18] employed features such as "$\mathbf{x}$ begins with a number", "$\mathbf{x}$ ends with a question mark", etc. Hence, MEMMs support long-distance interactions.

The IOHMM is similar to the MEMM except that it introduces hidden state variables s_t in addition to the output labels y_t. Sequential interactions are modeled by the s_t variables. To handle these hidden variables during training, the

Expectation-Maximization (EM; [6]) algorithm is applied. Bengio and Frasconi [2] report promising results on various artificial sequential supervised learning and sequence classification problems.

Unfortunately, the MEMM and IOHMM models suffer from a problem known as the *label bias problem*. To understand the origins of the problem, consider the MEMM and note that

$$\sum_{y_t} P(y_t|y_{t-1}, \mathbf{x}_1, \ldots, \mathbf{x}_t) = \sum_{y_t} P(y_t|y_{t-1}, \mathbf{x}_t) \cdot P(y_{t-1}|\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$$
$$= 1 \cdot P(y_{t-1}|\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}) = P(y_{t-1}|\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$$

This says that the total probability mass "received" by y_{t-1} (based on $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$) must be "transmitted" to labels y_t at time t regardless of the value of $\mathbf{x}_t$. The only role of $\mathbf{x}_t$ is to influence *which* of the labels receive more of the probability at time t. In particular, *all* of the probability mass *must* be passed on to some y_t even if $\mathbf{x}_t$ is completely incompatible with y_t.

For example, suppose that there are two labels $\{1, 2\}$ and that the input string $\mathbf{x} =$"rob" is supposed to get the label string "111" and $\mathbf{x} =$"rib" is supposed to get the label string "222". Consider what happens with the input string $\mathbf{x} =$"rib". After observing the $\mathbf{x}_1 = r$, the probability of y_1 is evenly split between labels "1" and "2": $P(y_1 = 1|\mathbf{x}_1 = r) = P(y_1 = 2|\mathbf{x}_1 = r) = 0.5$. After observing $\mathbf{x}_2 = i$, the probability remains equally split, because the 0.5 probability for $P(y_1 = 1|\mathbf{x}_1 = r)$ must be passed on to $P(y_2 = 1|\mathbf{x}_1 = r, \mathbf{x}_2 = i)$, since the $y_1 = 1 \rightarrow y_2 = 2$ transition has probability 0. After observing $\mathbf{x}_3 = b$, the probability of $y_3 = 1$ and $y_3 = 2$ remains equally split. So the MEMM has completely ignored the "i"! The same problem occurs with the hidden states s_t of the IOHMM.

3.4 Conditional Random Fields

Lafferty, McCallum, and Pereira [15] introduced the conditional random field (CRF; Figure 2(d)) to try to overcome the label bias problem. In the CRF, the relationship among adjacent pairs y_{t-1} and y_t is modeled as an Markov Random Field *conditioned on* the $\mathbf{x}$ inputs. In other words, the way in which the adjacent y values influence each other is determined by the input features.

The CRF is represented by a set of *potentials* $M_t(y_{t-1}, y_t|\mathbf{x})$ defined as

$$M_t(y_{t-1}, y_t|\mathbf{x}) = \exp\left(\sum_\alpha \lambda_\alpha f_\alpha(y_{t-1}, y_t, \mathbf{x}) + \sum_\beta \lambda_\beta g_\beta(y_t, \mathbf{x})\right),$$

where the f_α are boolean features that encode some information about y_{t-1}, y_t, and arbitrary information about $\mathbf{x}$, and the g_β are boolean features that encode some information about y_t and $\mathbf{x}$. As with MEMM's and IOHMM's, arbitrarily long-distance information about $\mathbf{x}$ can be incorporated into these features.

As with HMM's, CRF's assume two special labels 0 and $K + 1$ to indicate the start and end of the sequence. Let $M_t(\mathbf{x})$ be the $(K + 2) \times (K + 2)$ matrix of potentials for all possible pairs of labels for y_{t-1} and y_t.

The CRF computes the conditional probability $P(\mathbf{y}|\mathbf{x})$ according to

$$P(\mathbf{y}|\mathbf{x}) = \frac{\prod_{t=1}^{L} M_t(y_{t-1}, y_t|\mathbf{x})}{\left[\prod_{t=1}^{L} M_t(\mathbf{x})\right]_{0,K+1}},$$

where L is one more than the length of the strings, $y_0 = 0$, $y_L = K+1$, and the denominator is the $(0, K+1)$ entry in the matrix product of the M_t potential matrices. The normalizer in the denominator is needed because the potentials M_t are unnormalized "scores".

The training of CRFs is expensive, because it requires a global adjustment of the λ values. This global training is what allows the CRF to overcome the label bias problem by allowing the $\mathbf{x}_t$ values to modulate the relationships between adjacent y_{t-1} and y_t values. Algorithms based on iterative scaling and gradient descent have been developed both for optimizing $P(\mathbf{y}|\mathbf{x})$ and also for separately optimizing $P(y_t|\mathbf{x})$ for loss functions that depend only on the individual labels.

Lafferty, et al. compared the performance of the HMM, MEMM, and CRF models on a part-of-speech tagging problem. For a basic configuration, in which the MEMM and CRF features were defined to provide the same information as the HMM, the error rates of the three methods were HMM: 5.69%, MEMM: 6.37%, and CRF: 5.55%. This is consistent with the hypothesis that the MEMM suffers from the label bias problem but the HMM and the CRF do not. Lafferty et al. then experimented with providing a few simple spelling-related features to the MEMM and CRF models, something that is impossible to incorporate into the HMM. The resulting error rates where MEMM: 4.81% and CRF: 4.27%. Even more dramatic results are observed if we consider only "out of vocabulary" words (i.e., words that did not appear in any training sentence): HMM: 45.99%, MEMM: 26.99%, CRF: 23.76%. The spelling-related features provide powerful information for describing out of vocabulary words, whereas the HMM must rely on default observation probabilities for these words.

3.5 Graph Transformer Networks

In a landmark paper on handwritten character recognition, LeCun, Bottou, Bengio, and Haffner [16] describe a neural network methodology for solving complex sequential supervised learning problems. The architecture that they propose is shown in Figure 3. A graph transformer network is a neural network that transforms an input graph into an output graph. For example, the neural network in the figure transforms an input graph, consisting of the linear sequence of $\mathbf{x}_t$, into an output graph, consisting of a collection of u_t values. Each $\mathbf{x}_t$ is a feature vector attached to an edge of the graph; each u_t is a pair of a class label and a score. The Viterbi transformer analyzes the graph of u_t scores and finds the path through the graph with the lowest total score. It outputs a graph containing only this path, which gives the predicted y_t labels.

The architecture is trained globally by gradient descent. In order to do this, each graph transformer must be differentiable with respect to any internal tunable parameters. LeCun et al. describe a method called *discriminative forward*

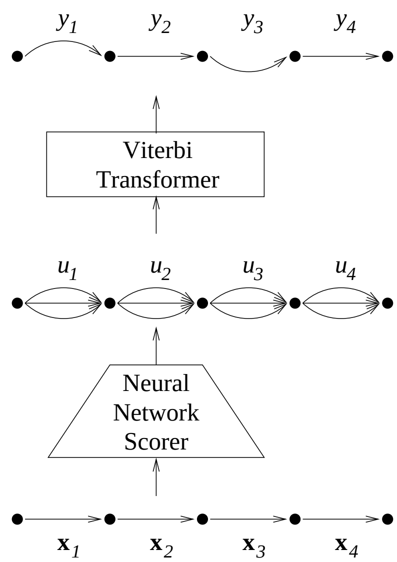

Fig. 3. The GTN architecture containing two graph transformers: a neural network and a Viterbi transformer.

training that adjusts the parameters in the neural network to reduce the score along paths in the u graph corresponding to the correct label sequence $\mathbf{y}$ and to increase the scores of the other paths. An advantage of this approach is that arbitrary loss functions can be connected to the output of the Viterbi transformer, and the network can be trained to minimize the loss on the training data.

4 Concluding Remarks

Sequential supervised learning problems arise in many applications. This paper has attempted to describe the sequential supervised learning task, discuss the main research issues, and review some of the leading methods for solving it. The four central research issues are (a) how to capture and exploit sequential correlations, (b) how to represent and incorporate complex loss functions, (c) how to identify long-distance interactions, and (d) how to make the learning algorithms fast. Our long-term goal should be to develop a toolkit of off-the-shelf techniques for sequential supervised learning. Although we are still some distance from this goal, substantial progress has already been made, and we can look forward to more exciting work in the near future.

References

1. G. Bakiri and T. G. Dietterich. Achieving high-accuracy text-to-speech with machine learning. In R. I. Damper, editor, *Data Mining Techniques in Speech Syn-*

thesis. Chapman and Hall, New York, NY, 2002.
2. Y. Bengio and P. Frasconi. Input-output HMM's for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249, September 1996.
3. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.
4. C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.
5. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.
6. A. P. Dempster, N. M. Laird, and D. B Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc.*, B39:1–38, 1977.
7. J.L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
8. T. Fawcett and F. Provost. Adaptive fraud detection. *Knowledge Discovery and Data Mining*, 1:291–316, 1997.
9. C. L. Giles, G. M. Kuhn, and R. J. Williams. Special issue on dynamic recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(2), 1994.
10. A. E. Hoerl and R. W. Kennard. Ridge regression: biased estimation of nonorthogonal components. *Technometrics*, 12:55–67, 1970.
11. M.I. Jordan. Serial order: A parallel distributed processing approach. ICS Rep. 8604, Inst. for Cog. Sci., UC San Diego, 1986.
12. Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.
13. Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *Proc. 13th Int. Conf. Machine Learning*, pages 284–292. Morgan Kaufmann, 1996.
14. Igor Kononenko, Edvard Šimec, and Marko Robnik-Šikonja. Overcoming the myopic of inductive learning algorithms with RELIEFF. *Applied Intelligence, 7*(1): 39–55, 1997.
15. John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Int. Conf. Machine Learning*, San Francisco, CA, 2001. Morgan Kaufmann.
16. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
17. Oded Maron and Andrew W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Adv. Neural Inf. Proc. Sys. 6*, 59–66. Morgan Kaufmann, 1994.
18. Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Int. Conf. on Machine Learning*, 591–598. Morgan Kaufmann, 2000.
19. Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
20. N. Qian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *J. Molecular Biology*, 202:865–884, 1988.
21. J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, 1993.
22. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing – Explorations in the Microstructure of Cognition*, chapter 8, pages 318–362. MIT Press, 1986.
23. T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce english text. *Journal of Complex Systems*, 1(1):145–168, February 1987.
24. A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. *Adv. Neural Inf. Proc. Sys. 3*, 875–882, Morgan Kaufmann, 1991.