Conditional Random Field Notes

Laura Ruis

January 15, 2019

Abstract

Notes on implementing a linear-chain CRF for POS tagging and a general CRF for dependency parsing. These implementations involve the forward-backward algorithm, Viterbi decoding and Tutte's Matrix Tree Theorem. The first two sections briefly go over the theory behind structured prediction and conditional random fields, and the final two sections go deeper into the implementation of these models for two tasks.

Contents

1 Structured Prediction

Structured prediction is a framework for solving problems of classification or regression in which the output variables are mutually dependent or constrained. It can be seen as a subset of prediction where we make use of the structure in the output space. The loss function used in structured prediction considers the output as a whole. If we denote the input by x, the output space (set of all possible outputs) by $\mathcal{Y}$ and $f(x, y)$ some function that expresses how well y fits x, then prediction can be denoted by the following formula:

$$y^* = arg \max_{y \in \mathcal{Y}} f(x, y)$$

The distinction between regular classification and structured prediction lies in the fact that for the latter case some kind of search needs to be done over the output space, namely finding the correct $y \in \mathcal{Y}$. An exhaustive search would usually be infeasible and structure prediction methods are needed.
In dependency parsing, x would correspond to a sentence, y to a dependency structure (tree) and $\mathcal{Y}$ to all possible dependency structures. A conditional random field (CRF) is a probabilistic model very useful for structured prediction.

2 Conditional Random Field

A conditional random field [3] is a probabilistic graphical model that combines advantages of discriminate classification and graphical models[6].

2.1 Discriminative vs. Generative

In the *discriminative* approach to classification we model $p(y \mid x)$ directly as opposed to the *generative* case where we model $p(x \mid y)$ and use it together with Bayes' rule for prediction. If we model the conditional directly, dependencies among x itself play no rule, resulting in a much simpler inference problem than modeling the joint $p(y, x)$. Generative models describe how some label y can generate some feature vector x, whereas discriminative models directly describe how to assign a feature vector x a label y.

2.2 Graphical Models

Graphical models represent complex distributions over many variables as a product of local factors of smaller subsets of variables. Based on what kind of graphical structure one assumes, one can use linear chain CRFs or more general CRFs. Generative models are often most naturally represented by directed graphical models ($p(x, y) = p(y)p(x \mid y)$), whereas discriminative models are most naturally represented by undirected graphical models. CRFs can be represented as *factor graphs*.
A factor graph is a graph $G = (V, F, E)$, in which V denotes the random variables and F denotes the factors. A factor graph describes how a complex probability distribution factorizes, and thus imposes independence relations. A distribution $p(\mathbf{y})$ factorizes according to a factor graph G if there exists a set of local functions ψ_a such that p can be written as:

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{a \in F} \psi_a(y_{N(a)})$$

The factors ψ_a are each only dependent on the neighbors y_i of a (denoted with $y_{N(a)}$) in the factor graph. The factors need to be larger than zero ($\psi_a(y_{N(a)}) \geq 0$), but need not have a probabilistic interpretation. The trick in using a factor graph for a task like classification is defining a set of feature functions that are nonzero only for a single class.

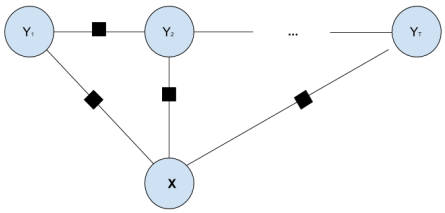

Figure 1: An instance of a linear-chain CRF

2.3 Linear-Chain CRF

A linear-chain CRF is simply a way of modeling the conditional probability distribution $p(\mathbf{y} \mid \mathbf{x})$ with a specific choice of feature functions and specific independence assumptions.
If we define $\boldsymbol{\theta}$ as parameters, $\mathcal{F} = \{f_k(y, y', \mathbf{x}_t)\}_{k=1}^{K}$ as a set of real-valued feature functions, a *linear-chain CRF* is conditional probability distribution that factorizes according to:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(x)} \prod_{t=1}^{T} \underbrace{\exp\left\{\sum_{k=1}^{K} \boldsymbol{\theta}_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}}_{\text{Factors in a FG: } \psi_t(y_t, y_{t-1}, \mathbf{x}_t)}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \boldsymbol{\theta}_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}$$

This is a factor graph with factors $\psi_t(y_t, y_{t-1}, \mathbf{x}_t)$, as shown in Figure 1. The dependence of each y_t on the entire input sequence is a modeling assumption. In the equation, each factor contains a vector $\mathbf{x}_t$ that can contain all information necessary for computing the features at time step t, which may include features from all other time steps. The graphical model underlying the particular linear-chain CRF in Figure 1 implies two independence assumptions:

1. Each state is only dependent on its immediate predecessor and x: $y_t \perp\!\!\!\perp \{y_1, \ldots, y_{t-2}\} \,|\, y_{t-1}, \mathbf{x}$
2. Each observation x_t only depends on the current state y_t: $x_t \perp\!\!\!\perp \{x_i\}_{i \neq t}, \{y_i\}_{i \neq t} \mid y_t, , \mathbf{x}$

2.4 Training a Linear-Chain CRF

A linear-chain CRF can be trained with maximum likelihood estimation. The log-likelihood over some dataset with N datapoints is:

$$\begin{aligned}
\log \mathcal{L}(\boldsymbol{\theta}) = \log \prod_{i=1}^{N} p(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}) &\overset{\text{FG}}{=} \log \left[\prod_{i=1}^{N} \frac{1}{Z(\mathbf{x}^{(i)})} \prod_{t=1}^{T} \psi_t(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)})\right] \\
&\overset{\text{CRF}}{=} \log \left[\prod_{i=1}^{N} \frac{1}{Z(\mathbf{x}^{(i)})} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \boldsymbol{\theta}_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)})\right\}\right] \\
&= \log \left[\prod_{i=1}^{N} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \boldsymbol{\theta}_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)})\right\}\right] + \log \left[\prod_{i=1}^{N} \frac{1}{Z(\mathbf{x}^{(i)})}\right] \\
&= \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \boldsymbol{\theta}_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} Z(\mathbf{x}^{(i)})
\end{aligned}$$

We can optimize this likelihood with gradient descent, and the gradients in the parameters involve computing the factors, meaning we can use probabilistic inference methods to compute them efficiently. Training a linear-chain CRF then comes down to two things:

- Inference of $p(y_t \mid \mathbf{x})$, $p(y_t, y_{t-1} \mid \mathbf{x})$ and $Z(\mathbf{x})$ (subroutines of parameter estimation, to see why refer to Section 2.6).
- Decoding: $\mathbf{y}^* = arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$

Inference in this model can be done with the forward-backward (belief propagation) algorithm, and decoding can be done with the Viterbi algorithm.

2.4.1 Forward-Backward Algorithm for a Linear-Chain CRF

The forward-backward algorithm defines an efficient way of calculating marginals from joints ($p(x) = \sum_y p(x, y)$)[1].

Finish from http://www.cs.columbia.edu/~mcollins/fb.pdf

2.4.2 Viterbi for a Linear-Chain CRF

2.5 General CRF

The general definition of a CRF:
Let G be a factor graph over $X$and Y, then (X, Y) is a conditional random field if for any value $\mathbf{x} \in X$, the distribution $p(\mathbf{y} \mid \mathbf{x})$ factorizes according to G.
The conditional distribution for a CRF is then:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{a=1}^{A} \psi_a(\mathbf{y}_a, \mathbf{x}_a)$$

Where the only difference with a general factor graph is the dependence of the partition function on the input $\mathbf{x}$. If we take log-linear factors (meaning that $\log(\psi_a(\mathbf{y}_a, \mathbf{x}_a))$ will be linear over a set of feature functions), the CRF becomes:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\psi_A \in \mathcal{F}} \exp\left\{ \sum_{k=1}^{K(A)} \boldsymbol{\theta}_{a,k} f_{a,k}(\mathbf{x}_a, \mathbf{y}_a) \right\}$$

2.6 Training a General CRF

The methods used to train a linear-chain CRF also apply to a general CRF. We can train it with maximum likelihood, and in order to do so we need the partition function, the marginals and an efficient way of decoding. There are many algorithms for efficient inference in graphical models, like the junction tree algorithm for *exact inference* and Monte Carlo methods or Variational Inference for *approximate inference*. In the case where we are dealing with complete graphs, we can use Tutte's *Matrix Tree Theorem* (MTT) to evaluate the marginals and partition function efficiently. The decoding problem can be solved with different algorithms based on the specification of the CRF.
The log likelihood for our general CRF is almost identical to the linear case, so we won't derive it again:

$$\begin{aligned}
\log \mathcal{L}(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}) \\
&= \sum_{i=1}^{N} \sum_{\psi_a \in \mathcal{F}} \sum_{k=1}^{K(A)} \boldsymbol{\theta}_{a,k} f_{a,k}(\mathbf{x}_a^{(i)}, \mathbf{y}_a^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)})
\end{aligned}$$

Calculating the loss function involves evaluating the partition function (if negative log-likelihood is used), which can be done efficiently with the MTT as described in Section 2.6.1. The partial

[1] Taken from http://www.cs.columbia.edu/~mcollins/fb.pdf

derivative w.r.t. the parameters is[2]:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_{p,t}} = \sum_{i=1}^{N} \sum_{\psi_a \in \mathcal{F}} f_{p,t}(\mathbf{x}_a^{(i)}, \mathbf{y}_a^{(i)}) - \sum_{i=1}^{N} \sum_{\psi_a \in \mathcal{F}} \sum_{\mathbf{y}' \in Y} f_{p,t}(\mathbf{x}_a^{(i)}, \mathbf{y}_a'^{(i)}) p(\mathbf{y}' \mid \mathbf{x}^{(i)})$$

Which involves the marginals $\sum_{\mathbf{y}' \in Y} p(\mathbf{y}' \mid \mathbf{x}^{(i)})$ that can be computed with the MTT for every data point.

2.6.1 The Matrix Tree Theorem

This section contains a brief summary of the lecture on the Matrix Tree Theorem by Mark Muldoorn[2]. Kirchoff's matrix tree theorem is a theorem from linear algebra that defines an efficient way to count spanning trees in an undirected graph, and Tutte extended it to count spanning trees in a directed graph, which is exactly what we need to calculate the partition function. Only the latter theorem is summarized here and for the proof the reader is referred to the original lecture.
A directed tree is a graph that contains a root and is a tree. A *root* is a vertex from which all other vertices are accessible.
A spanning tree of a directed tree is a graph where each vertex has at most one incoming edge. All vertices are covered with a minimum number of edges, there are no cycles and the graph is not disconnected. Every connected, undirected graph has at least one spanning tree.
Tutte's Directed Matrix Tree Theorem:
If $G(V, E)$ is a directed graph with vertices $V = \{v_1, \ldots, v_n\}$ and L is the Laplacian matrix whose entries are given by:

$$L_{ij} = \begin{cases} \deg_{\text{in}}(v_j) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

The Laplacian matrix L can also be found by $L = D - A$, where D is a diagonal matrix with entries $deg_{in}(v_i)\ \forall\ v_i \in V$ and A the adjacency matrix with entries 1 for adjacent vertices (if $i \neq j$ and $(v_i, v_j) \in E$) and 0 otherwise (0 on the diagonal).
Then the number of spanning trees N_j with a root at v_j is:

$$N_j = \det\left(\hat{L}_j\right)$$

Where $\hat{L}_j$ is the matrix obtained by removing row and column j.
Now if $y \in \mathcal{Y}$ is a directed graph with vertices $y_1, \ldots, y_T$, the partition function $Z(\mathbf{x})$ of the conditional probability distribution over each $\mathbf{x}$ is exactly the number of existing spanning trees for a particular $\mathbf{x}$: $N_1 = Z(\mathbf{x})$ with $G(V, E) = y$ and $\mathbf{x}$ the corresponding features. This result conveniently means we can utilize the MTT to calculate the partition function for training a general CRF with maximum likelihood.

3 Part-of-Speech Tagging with a Linear-Chain CRF

[4]

4 Dependency Parsing with a General CRF

In the paper Structured Prediction Models via the Matrix-Tree Theorem by Koo et al. [2], the authors describe very thoroughly how a CRF can be used for dependency parsing. The information in this section is based on their work. The next section will briefly define the problem of non-projective, arc-factored, graph-based dependency parsing, with most information taken from Jurafsky and Martin [1] and notation taken from [2]. Section 4.2 will then describe how to use a CRF for this task.

[2]Taken from https://personalpages.manchester.ac.uk/staff/mark.muldoon/Teaching/DiscreteMaths/LectureNotes/IntroToMatrixTree.pdf

4.1 Dependency Parsing

In dependency-based approaches to syntax, the structure of a sentence is described in terms of a set of binary relations that hold between the words in a sentence [1]. Larger notions of constituency are not directly encoded in dependency analyses. Dependency parsing is very useful for languages with free word order. Where other types of grammars would need separate rules for each place certain phrases would occur, dependency-based approaches would just have a link for this relation. Thus, a dependency grammar approach abstracts away from word-order information, representing only the information that is necessary for the parse. Relations provide approximation to semantic relation between predicates, which is useful in many domains. Constituent-based approaches to parsing provide similar information, but it often has to be distilled from the trees via techniques such as the head finding rules.

Graph-based parsing defines a search through the space of possible trees for a sentence with high score and constructs trees from it with algorithms from graph theory. The *edge-factored approach* simplifies this problem by assuming the score of a tree is the sum of its parts (edges). Graph-based parsing is the preferred approach for non-projective languages, since it can handle longer range dependencies, whereas *transition based parsing* makes greedy local decisions.

In the case of non-project dependency parsing, a parse corresponds to a directed spanning tree on a complete directed graph of T nodes, where T is the length of the sentence [5]. Now we define the following:

- $\mathbf{x}$ a sentence with T words.
- $\mathbf{y}$ a dependency parse consisting of head-modifier pairs (h, m) with $h \in [0, \ldots, T]$ in $m \in [1, \ldots, T]$ the indices of the respective head and modifier in the sentence $\mathbf{x}$ ($h = 0$ indicates the root).
- $D(\mathbf{x}) = \{(h, m) : h \in [0, \ldots, T], m \in [1, \ldots, T]\}$ all possible dependencies for $\mathbf{x}$.
- $\mathcal{Y}(\mathbf{x})$ the space of all possible parses for $\mathbf{x}$
- $\boldsymbol{\theta}$ a vector with a score $\theta_{h,m} \forall (h, m) \in D(\mathbf{x})$
- $\mathbf{r}(\boldsymbol{\theta})$ a vector of root scores indicating how likely each word in $\mathbf{x}$ is the modifier of the root symbol (with $r_m(\boldsymbol{\theta}) = \theta_{0,m}$[3]).
- G a complete, directed graph on T nodes, where each node is a word in $\mathbf{x}$, and each edge a dependency between two words.

4.2 Discriminative Dependency Parsing for Single-Root Trees

We can use a general CRF to do graph-based dependency parsing. Recall from Section 2.5 the definition of a CRF:

$$p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}, \boldsymbol{\theta})} \prod_{\psi_A \in \mathcal{F}} \exp \left\{ \sum_{k=1}^{K(A)} \boldsymbol{\theta}_{a,k} f_{a,k}(\mathbf{x}_a, \mathbf{y}_a) \right\}$$

Now we plug in features that describe relation (h, m) in $\mathbf{x}$ and a weight vector $\mathbf{w}$.

$$= \frac{1}{Z(\mathbf{x}, \boldsymbol{\theta})} \prod_{(h,m) \in \mathbf{y}} \exp \left\{ \underbrace{\mathbf{w} \cdot f(h, m, \mathbf{x})}_{\text{weighted edge}} \right\}$$

[3]These scores are part of $\boldsymbol{\theta}$, but are defined separately as well because to be able to use the MTT we need a complete directed graph and in a correct parse the root will have no incoming edge. This means the root has to be addressed separately.

Now we define $\theta_{h,m} = \mathbf{w} \cdot f(h, m, \mathbf{x})$. The scores are only dependent on the dependency relation and the entire sentence. We separate out the root scores to get our final CRF equation.

$$= \frac{1}{Z(\mathbf{x}, \boldsymbol{\theta})} \exp \left\{ r_{\text{root}(\mathbf{y})}(\boldsymbol{\theta}) + \sum_{(h,m) \in y : h \neq 0} \theta_{h,m} \right\}$$

The dependence of $\theta_{h,m}$ on $\mathbf{x}$ is left out.

The *partition function* is then:

$$Z(\mathbf{x}, \boldsymbol{\theta}) = \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \exp \left\{ r_{\text{root}(\mathbf{y})}(\boldsymbol{\theta}) + \sum_{(h,m) \in y : h \neq 0} \theta_{h,m} \right\}$$

As mentioned before, training a general CRF comes down to three inference problems:

- *Decoding*: $y^* = \arg\max_{y \in \mathcal{Y}(\mathbf{x}, \boldsymbol{\theta})} p(\mathbf{y} \mid \mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \sum_{(h,m) \in \mathbf{y}} \theta_{h,m}$, by the arc-factored assumption and CRF (the exponent is a linear transformation and the denominator is the same for any $\mathbf{y} \in \mathcal{Y}$). Can be done with CLE (NEEDS CITATION) algorithm for the non-projective case, non discussed further in this work.
- *Calculate the partition function*: $Z(\mathbf{x}, \boldsymbol{\theta})$. Needed for the loss function.
- *Calculate the marginals*: $p(h, m \mid \mathbf{x})$. Necessary for the gradients, but can be done with automatic differentiation features of deep learning libraries, so not mentioned further.

4.2.1 Calculating the Partition Function

As described in Section 2.6.1, calculating the partition function can be done with Tutte's Directed Matrix Tree Theorem. For graph-based dependency parsing we need a couple of adjustments. First of all, we need the sum of *weighted* directed spanning trees. The weighted adjacency matrix for this type of graph can be defined as follows:

$$A_{h,m}(\boldsymbol{\theta}) = \begin{cases} 0 & \text{if } h = m \\ \exp\left(\theta_{h,m}\right) & \text{otherwise} \end{cases}$$

The Laplacian $L(\boldsymbol{\theta}) \in \mathcal{R}^{T \times T}$ is then:

$$L_{h,m} = \begin{cases} \sum_{h'=1}^{T} A_{h',m}(\boldsymbol{\theta}) & \text{if } h = m \\ -A_{h,m}(\boldsymbol{\theta}) & \text{otherwise} \end{cases}$$

The second adjustment stems from the fact that the graph G does not include the root symbol, but we do need to know which word in $\mathbf{x}$ is the modifier of the root. Using the theorem we can calculate the partition function as follows, separately accounting for the root scores:

$$Z(\mathbf{x}, \boldsymbol{\theta}) = \sum_{m=1}^{T} r_m(\boldsymbol{\theta}) \det(\hat{L}_m(\boldsymbol{\theta}))$$

But this needs T determinant calculations. Koo et al. adjusted this to a calculation where only one determinant needs to be evaluated. The proof for this method can be found in their paper. Define a new matrix, where the first row of the Laplacian is replaced by the root scores:

$$\bar{L}_{h,m}(\boldsymbol{\theta}) = \begin{cases} r_m(\boldsymbol{\theta}) & \text{if } h = 1 \\ L_{h,m}(\boldsymbol{\theta}) & \text{otherwise} \end{cases}$$

Then according to proposition 1 in [2], we can calculate the partition function efficiently as follows:

$$Z(\mathbf{x}, \boldsymbol{\theta}) = \det(\bar{L}(\boldsymbol{\theta}))$$

Which is all we need to train a general CRF for graph-based dependency parsing.

References

[1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[2] Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. Structured prediction models via the matrix-tree theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[3] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[4] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354, 2016.

[5] Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 91–98, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[6] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Found. Trends Mach. Learn.*, 4(4):267–373, April 2012.