SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.040 Natural Language Processing, Summer 2020

Supplemental Notes on
Conditional Random Fields

Last update: Saturday 13th June, 2020 22:15

1 Discriminative Learning for Structured Prediction

In the machine learning course, as well as the earlier classes of this course, we have discussed how to use a generative model for solving the structured prediction problem - hidden Markov models (HMM). Recall in HMM, we were interested in the following:

$$p(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$

where $\boldsymbol{x}$ is the input sequence and $\boldsymbol{y}$ is the output sequence.

This formulation is also used in the naive Bayes model, in which case the input is a vector and the output is a discrete label. As we have discussed, there are typically discriminative counterparts of such generative models. Specifically, for naive Bayes, the discriminative counterpart is the logistic regression (or softmax regression, if $\boldsymbol{y}$ can take more than two values). The discriminative counterpart of the hidden Markov model is the linear-chain conditional random fields (CRF).

In conditional random fields, we are interested in the following:

$$p(\boldsymbol{y}|\boldsymbol{x}) \tag{2}$$

Now here comes the first question: how do we parameterize such a model?

1.1 Entropy

Before we discuss the parameterization, let us get ourselves familiar with the concept of entropy. What is entropy? The entropy is used for measuring the degree of uncertainty. It is defined as follows. Detailed motivation of the form has already been discussed in class.

$$\mathbf{H}(p(x)) = -\sum_{x} p(x) \log p(x) \tag{3}$$

Now, our idea here is to maximize the degree of uncertainty of our conditional model above, while making sure we respect our training data. That means, we shall make sure certain constraints are satisfied. What are the constraints?

First of all, we may assume that there are some features $\mathbf{f}(\boldsymbol{x}, \boldsymbol{y}) = [f_1(\boldsymbol{x}, \boldsymbol{y}), \ldots, f_K(\boldsymbol{x}, \boldsymbol{y})]$ that can be used to characterize the quality of the $(\boldsymbol{x}, \boldsymbol{y})$ tuple.

Assume we have a training set consisting of training instances $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ (for $i = 1, \ldots, N$). Let's use $\boldsymbol{x}$ to denote the list $\langle \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \rangle$, and use $\boldsymbol{y}^*$ to denote $\langle \boldsymbol{y}_1, \ldots, \boldsymbol{y}_N \rangle$ (however, later I may use $\boldsymbol{x}$ and $\boldsymbol{y}$ to refer to a single input and output sequence for one instance; this should be clear from its context). Our first set of constraints would be as follows:

$$\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] = \mathbf{E}_{\hat{p}(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] \tag{4}$$

for $j = 1 \ldots, K$, where $p(\boldsymbol{y}|\boldsymbol{x})$ is our model, and $\hat{p}(\boldsymbol{y}|\boldsymbol{x})$ is the empirical conditional probability calculated from data. However, as we mentioned during class, this distribution is essentially a boring distribution, which assigns probability 1 to the single observed list of output structures $\boldsymbol{y}^*$ and 0 to all other list of structures.

This means, the constraints can be written as:

$$\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] = f_j(\boldsymbol{x}, \boldsymbol{y}^*) \tag{5}$$

Note that we can equivalently write the above as:

$$\sum_i \mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x}_i)}[f_j(\boldsymbol{x}_i, \boldsymbol{y})] = \sum_i f_j(\boldsymbol{x}_i, \boldsymbol{y}_i) \tag{6}$$

These constraints say that we would like to design such a model that, for each feature f_j, when we calculate the expected number of times it appears in the training set based on our model, its value should be exactly the same as that of the empirical count (i.e., the actual number of times we see the feature in our training set).

The other constraint would be:

$$\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) = 1 \tag{7}$$

Now the problem becomes:

$$\min_{p(\boldsymbol{y}|\boldsymbol{x})} -\mathbf{H}(p(\boldsymbol{y}|\boldsymbol{x})) \tag{8}$$

subject to the constraints

$$\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] = \mathbf{E}_{\hat{p}(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] \tag{9}$$

$$\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) = 1 \tag{10}$$

Let us form the Lagrangian:

$$\min_{p(\boldsymbol{y}|\boldsymbol{x})} \max_{\lambda} \Lambda(p, \lambda) \tag{11}$$

where $\Lambda(p, \lambda)$ is defined as follows:

$$-\mathbf{H}(p(\boldsymbol{y}|\boldsymbol{x})) - \sum_{j=1}^{K} \lambda_j \Big(\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] - f_j(\boldsymbol{x}, \boldsymbol{y}^*) \Big) - \lambda_0 \left(\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) - 1 \right) \tag{12}$$

where $\lambda_j \in (-\infty, +\infty)$.

How do we solve this min-max problem? A standard first step is to make it a max-min problem:

$$\max_{\lambda} \min_{p(\boldsymbol{y}|\boldsymbol{x})} \Lambda(p, \lambda) \tag{13}$$

Next, we can first focus on the inner optimization problem, and try to see what should be the optimal values for $p(\boldsymbol{y}|\boldsymbol{x})$, assuming λ is given:

$$\min_{p(\boldsymbol{y}|\boldsymbol{x})} \Lambda(p, \lambda) = \min_{p(\boldsymbol{y}|\boldsymbol{x})} -\mathbf{H}(p(\boldsymbol{y}|\boldsymbol{x})) - \sum_{j=1}^{K} \lambda_j \Big(\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] - f_j(\boldsymbol{x}, \boldsymbol{y}^*) \Big) - \lambda_0 \left(\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) - 1 \right) \tag{14}$$

Now, let us take the partial derivative with respect to $p(\boldsymbol{y}|\boldsymbol{x})$ and set it to zero, yielding:

$$\frac{\partial \Lambda(p, \lambda)}{\partial p(\boldsymbol{y}|\boldsymbol{x})} = \log p(\boldsymbol{y}|\boldsymbol{x}) + 1 - \sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}) - \lambda_0 = 0 \tag{15}$$

This yields:

$$\log p(\boldsymbol{y}|\boldsymbol{x}) = -(1 - \lambda_0) + \sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}) \tag{16}$$

This give us:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \exp \left(\sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}) \right) / \exp(1 - \lambda_0) \tag{17}$$

From here we arrive at the parametric form for the conditional probability as follows:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z} \exp \left(\sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}) \right) \tag{18}$$

As we can see, the logistic regression is a special case of such a model.

Ok, now, let's see how we can simplify the inner optimization problem:

$$-\mathbf{H}(p(\boldsymbol{y}|\boldsymbol{x})) - \sum_{j=1}^{K} \lambda_j \Big(\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] - f_j(\boldsymbol{x}, \boldsymbol{y}^*) \Big) - \lambda_0 \left(\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) - 1 \right) \tag{19}$$

The last term is zero, based on Eq 17, yielding:

$$-\mathbf{H}(p(\boldsymbol{y}|\boldsymbol{x})) - \sum_{j=1}^{K} \lambda_j \Big(\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_j(\boldsymbol{x}, \boldsymbol{y})] - f_j(\boldsymbol{x}, \boldsymbol{y}^*) \Big) \tag{20}$$

$$= -\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \log p(\boldsymbol{y}|\boldsymbol{x}) - \sum_{j=1}^{K} \lambda_j \Big(\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) f_j(\boldsymbol{x}, \boldsymbol{y}) - f_j(\boldsymbol{x}, \boldsymbol{y}^*) \Big) \tag{21}$$

$$= -\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \log p(\boldsymbol{y}|\boldsymbol{x}) - \sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}) + \sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}^*) \tag{22}$$

$$= -\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \Big(-(1 - \lambda_0) \Big) + \sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}^*) \tag{23}$$

$$= -(1 - \lambda_0) + \sum_{j=1}^{K} \lambda_j f_j(\boldsymbol{x}, \boldsymbol{y}^*) \tag{24}$$

$$= \log p(\boldsymbol{y}^*|\boldsymbol{x}) \tag{25}$$

So in other words, we just need to solve the following problem now:

$$\max_{\lambda} \log p(\boldsymbol{y}^*|\boldsymbol{x}) \tag{26}$$

Recall that $\boldsymbol{x}$ and $\boldsymbol{y}^*$ gives us the training set. So this is the conditional log-likelihood defined over the training set. To maximize this, is equivalent to the minimization of the negative log-likelihood, as discussed next.

1.2 Objective and Gradient

As discussed, the loss function to be minimized should be:

$$\mathcal{L}(\mathbf{w}) = -\sum_{i} \log p(\boldsymbol{y}_i|\boldsymbol{x}_i) = -\sum_{i} \log \frac{\exp(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}_i) \cdot \mathbf{w})}{\sum_{\boldsymbol{y}} \exp(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \mathbf{w})} \tag{27}$$

$$= -\sum_{i} \left[\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}_i) \cdot \mathbf{w} - \log \sum_{\boldsymbol{y}} \exp(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \mathbf{w}) \right] \tag{28}$$

where $\mathbf{w} = [\lambda_1, \ldots, \lambda_K]$.

Now let us take the partial derivative with respect to λ_k:

$$\begin{aligned}
\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \lambda_k} &= \frac{\partial}{\partial \lambda_k} - \sum_i \log p(\boldsymbol{y}_i|\boldsymbol{x}_i) \\
&= \sum_i -\frac{\partial}{\partial \lambda_k} \log \frac{\exp\left(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}_i) \cdot \mathbf{w}\right)}{\sum_{\boldsymbol{y}} \exp\left(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \mathbf{w}\right)} \\
&= \sum_{i=1}^{n} \left(\frac{\partial}{\partial \lambda_k} \log \sum_{\boldsymbol{y}} \exp\left(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \mathbf{w}\right) - \frac{\partial}{\partial \lambda_k} (\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}_i) \cdot \mathbf{w}) \right) \\
&= \sum_i \left(\sum_{\boldsymbol{y}} \frac{\exp\left(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}) \cdot \mathbf{w}\right) f_k(\boldsymbol{x}_i, \boldsymbol{y})}{\sum_{\boldsymbol{y}''} \exp\left(\mathbf{f}(\boldsymbol{x}_i, \boldsymbol{y}'') \cdot \mathbf{w}\right)} - f_k(\boldsymbol{x}_i, \boldsymbol{y}_i) \right) \\
&= \sum_i \mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x}_i)}[f_k(\boldsymbol{x}_i, \boldsymbol{y})] - \sum_i f_k(\boldsymbol{x}_i, \boldsymbol{y}_i)
\end{aligned} \tag{29}$$

As we can see, this is exactly one of the constraints that we used in the optimization problem mentioned in the previous section (Eq 6). Our goal in the learning or optimization process is to make this zero, so that this constraint is satisfied.

Since both $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequences, how do we do learning efficiently? We will have to define the features locally, and then we will use the forward-backward algorithm to calculate the expected counts. Decoding can be done with the Viterbi algorithm.

2 Forward algorithm

Let us first consider the problem of calculating the objective function in Eq 28. The key question is how to calculate the second term in the brackets. We will make use of the forward algorithm for doing so.

Let us consider the following input sentence $\boldsymbol{x}$ = "I love NLP" with the output sequence $\boldsymbol{y}$ = $\mathbf{N}$ $\mathbf{D}$ $\mathbf{N}$. What shall the function $\mathbf{f}(\boldsymbol{x}, \boldsymbol{y})$ return? What we can do is to consider the very simple features as those used in the HMM. If we do so, the features involved in this input-output sequence pair would be (here I am using the same string representation as mentioned in our NLP project):

emission:N+I
emission:D+love
emission:N+NLP
transition:START+N
transition:N+D
transition:D+N
transition:N+STOP

If we assign each feature a weight, then we should be able to calculate the score associated with this particular input-output sequence pair. For example, if we have:

Feature	Weight
emission:N+I	$+1.0$
emission:D+love	$+2.0$
emission:N+NLP	-1.0
transition:START+N	-0.2
transition:N+D	$+7.0$
transition:D+N	-1.8
transition:N+STOP	$+9.0$

The score here should be the sum of all the weights of the involved features above. In this case it is 16. Let us call this way of calculating the weight for a particular output sequence "Method I".

It is equivalent to say the table is like the following:

Feature	Exp-Weight
emission:N+I	$\exp(+1.0)$
emission:D+love	$\exp(+2.0)$
emission:N+NLP	$\exp(-1.0)$
transition:START+N	$\exp(-0.2)$
transition:N+D	$\exp(+7.0)$
transition:D+N	$\exp(-1.8)$
transition:N+STOP	$\exp(+9.0)$

In this case, the score will be calculated as the multiplication of the exp-weights above. In this case it is $\exp(16)$. Let us call this way of calculating the score as "Method II".

Now let us try to understand the second term in Eq 28 better. What is it? It is essentially the log of the sum of the scores of all possible output sequences calculated using "Method II".

Now is it clear to you? How do we calculate the forward score? What we can do is simple. (We assume we define the emission and transition features in the same way as HMM.)

1. We first convert each feature weight w in the original dictionary into $\exp(w)$.
2. Next, we simply run the forward algorithm that we have learned in the ML course.
3. Finally, we retrieve the final score stored in the right-most node $\alpha_{\texttt{STOP}}(n+1)$, and return $\log \alpha_{\texttt{STOP}}(n+1)$.

Of course, this is the most straightforward way to implement the forward algorithm. Can you think of some other ways to do so? Do you remember in Part 3 of the ML design project, I mentioned if you encounter potential numerical overflow/underflow issue, you may need to find a way to address that issue? In fact, you may also potentially encounter such an issue when convertinig w into $\exp(w)$ or when summing them up. Can you think of some way to fix this limitation? Is it possible to define the α terms differently (e.g., in the log space, or, instead of storing $\alpha_u(k)$, let us perhaps store $\log \alpha_u(k)$ at each node)? Hopefully the second term in Eq 28 gives you some ideas.

3 Backward algorithm

To calculate the gradients, we will also need the backward scores.

We can follow a similar idea to calculate the backward scores.

1. We first convert each feature weight w in the original dictionary into $\exp(w)$.
2. Next, we simply run the backward algorithm that we have learned in the ML course.

How do we calculate a gradient for a feature? Let's consider the above-mentioned example input sentence and this feature "emission:N+I". Let's call this feature f_{123}:

$$\mathbf{E}_{p(\boldsymbol{y}|\boldsymbol{x})}[f_{123}(\boldsymbol{x}_i, \boldsymbol{y})] = \sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}_i) f_{123}(\boldsymbol{x}_i, \boldsymbol{y})$$

Here this function $f_{123}(\boldsymbol{x}_i, \boldsymbol{y})$ will basically return the number of times we have the feature "emission:N+I" based on the given input-output sequence pair $(\boldsymbol{x}_i, \boldsymbol{y})$. For example, if $\boldsymbol{y} =$ `START`$\mathbf{N}$ $\mathbf{I}$ $\mathbf{N}$ `STOP`, it will return 1. You may realize now it is similar to the calculation of the expected number of times we see a transition pattern used in the HMM model (soft EM). I will now leave it to you to figure out how to finish the last step of calculation of this expected (feature) count based on the forward and backward scores that you calculated. You may also think about how to perform this step efficiently in your implementation. Note also that these scores are calculated based on one instance, but the gradient is defined at the corpus level.

4 Viterbi algorithm

Similarly, the Viterbi algorithm here will be as follows:

1. We first convert each feature weight w in the original dictionary into $\exp(w)$.
2. Next, we simply run the Viterbi algorithm that we have learned in the ML course.