University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 14: October 16

Lecturer: Alexander Schwing Scribe: Jiahao Lin

1 Goals

- Learning of structured distributions
- Feature vectors for structured output space data
- Learning with structured output space data

Reading materials: D. Koller and N. Friedman; Probabilistic Graphical Models: Principles and Techniques;

2 Introduction

This is a recap lecture in a sense we are combing everything we have learned so far, and then we are gonna see a reasonably general framework that allows us to deduce everything which we have covered so far. In a sense that everything that we talked about in the past you can think of it as being represented in one of those formulas. This is lecture is about how we gonna learn in structured framework. Questions from last lecture: how do we know how to combine unary potentials and pair-wise potentials in structured models, how do we know what weight to give each of them. These are exactly the answers that we are trying to answer today. Let me try to get you guys in the same page, so that you will have the same picture in mind. One of the examples we went through last time: you are given a bunch of letters, trying to find what word does those four letters represent (word classification problem). We used unary potentials which are local evidence for every letter, we also have some form of pair-wise evidence which tell us the relation between two words. How much should we trust unary evidence and how much should we trust pair-wise evidence. How to combine them. This is the question we want to address today, we want to learn how much should we trust those evidence. We are shifting from inference to learning.

3 Recap

3.1 General framework for leaning:

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + F(\mathbf{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\mathbf{w}, x^{(i)}, y^{(i)}) \tag{1}$$

where $\frac{C}{2} \|\mathbf{w}\|_2^2$ is the regularization term, $F(\mathbf{w}, x^{(i)}, y^{(i)})$ is the score of ground truth, and ϵ is some positive constant. $L(y^{(i)}, \hat{y})$ is called task loss, which represents the difference between the predicted configuration and ground truth configuration. We never assume the task loss is differentiable in any form. The task loss could be the 0-1 loss because we dont need to differentiate with respect to its parameters and in face it doesnt have any parameter. And $F(\mathbf{w}, x^{(i)}, \hat{y})$ is the score of predicted configuration at configuration y hat, summing over all possible configurations. When ϵ is 1, we end up with log-loss of cross entrop. And epsilon is 0, we end up with hinge-loss.

From this model, how can we get to:

- Logistic regression? **Epsilon equals to 1.**
- Binary SVM? **Epsilon equals to 0.**
- Deep Learning? **Scoring function F becomes a composite function.**

3.2 Inference (prediction):

$$y^* = \arg\max_{\hat{\text{y}}} F(\mathbf{w}, x, y) \tag{2}$$

In binary classification case, we have one vector, and then we multiply this vector with our weight vector and corresponding feature vector, then we look at the number: positive predict it one class, negative predict another class. For multiclass regression type problem, inference is easy: we have a matrix of vectors, and a hand-crafted feature vector, we multiply the matrix with the feature vector and pick the class with highest score. Or use soft-max to transform the score into probability and then pick the highest probability. Whats the issue of this configuration? The search space is becoming too large, very inefficient.

$$\mathbf{y}^* = \arg\max_{\hat{\mathbf{y}}} \sum_r f_r(\mathbf{w}, x, \hat{\mathbf{y}}_r) \tag{3}$$

Solution: assume function F could be decompose into sum of terms, each of which depends on a subset that we care about. e.g. assume F could be decompose into a tree structure and then we can apply dynamic programming.

Inference algorithms:

- Exhaustive search, easiest but very slow, not feasible when search space is large.
- Dynamic programming, only works for tree structure.
- Formulate this task as a Integer linear program, still somewhat expensive.
- Relax it and end up with linear programming relaxation.
- Message passing, another form of dynamic programming.

4 Complex learning objects

4.1 Fit complex learning objects into the equation (1)

Start with a simpler example, linear multi-class formulation: instead of having a general function $F(\mathbf{w}, x, y)$, we have this inner product between a weight vector $\mathbf{w}$ and some hand-crafted feature vector $\psi(x, y)$. Let's take a look at the example of semantic segmentation.

In this picture, we want to figure out where are the bikes, where are the humans, and where is background and the edge. These are the tasks that we are interested in. Output space: say 1 mega pixel image, background class, human class, and bicycle class, three classes, so its $3^{\#pixels}$. Exhaustive search would be very inefficient. The model is often applied in this case is grid graphical model. Variable y_i is the variable of each pixel which can take three states. And pair-wise function $y_{i,j}$ which tell us information between y_i and y_j. How to trade off smoothness with respect to local evidence?

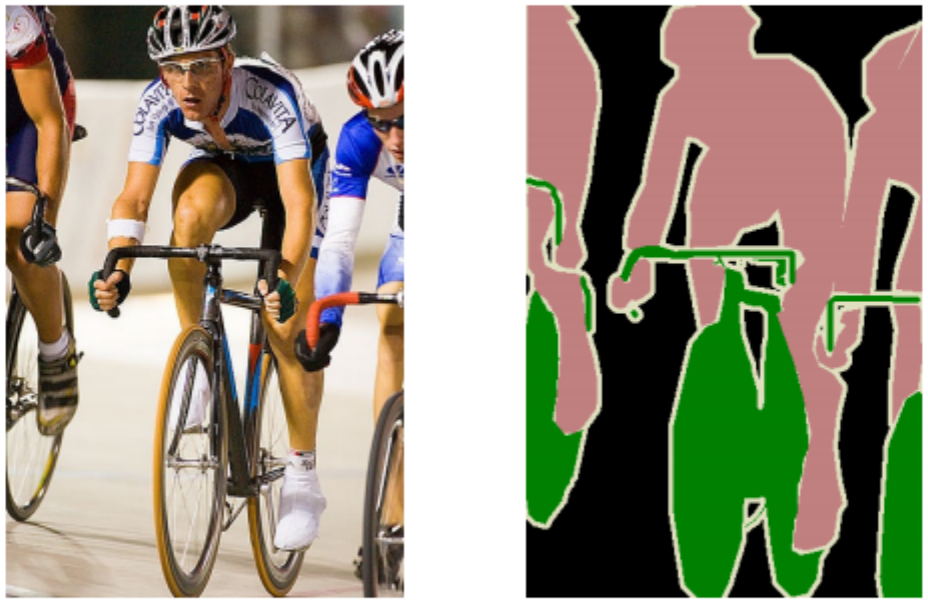

Figure 1: Image example

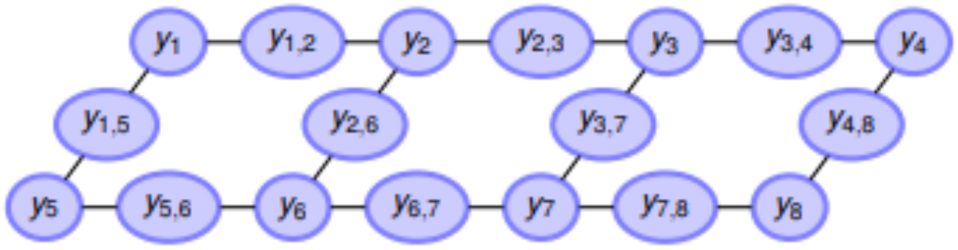

Figure 2: Grid model

Feature vector for structured objects:

$$f(x^{(i)}, \mathbf{y}) = \begin{bmatrix} \underline{f_1(x^{(i)}, \mathbf{y})} \\ \vdots \\ \overline{f_M(x^{(i)}, \mathbf{y})} \end{bmatrix} = \begin{bmatrix} \underline{\sum_r f_{1,r}(x^{(i)}, \mathbf{y_r})} \\ \vdots \\ \overline{\sum_r f_{M,r}(x^{(i)}, \mathbf{y_r})} \end{bmatrix} \tag{4}$$

Feature vector [2] is composed out of a set of numbers as before, but differently, we are now computing the number f1 using a sum of local terms. Simply plug in a configuration, and we can get a number out of it. After writing down in this summation form, lets come back to the earlier example that we want to weight unary evidence with respect to pair-wise evidence, how to plug into this framework? All of the unary evidence as one entry, and all of the pair-wise evidence as another entry. Given a configuration, you can compute the score for your pair-wise evidence as well as local evidence. The next step is to combine both local and pair-wise evidence into a single model. How? We just weight them with a vector w and compute the inner product of a weight vector that is trainable and a feature vector that contains some more complex functions [1].

$$F(\mathbf{w}, x^{(i)}, \mathbf{y}) = \mathbf{w}^\top f(x^{(i)}, \mathbf{y}) = \sum_{m=1}^{M} \mathbf{w}_m \left(\sum_r f_{m,r}(x^{(i)}, \mathbf{y}_r) \right) = \sum_r \hat{f}_r(\mathbf{w}, x^{(i)}, \mathbf{y}) \tag{5}$$

What we do in equation (5) is simply write out all the terms, becomes sum of individual models in each of the entries. Then exchange the summation to obtain the new scoring function. What we do here in this example is that we put all the unary evidence into a single entry and all of pair-wise evidence into another entry. We can also put y1 into the first entry, y2 into the second entryand y2,4 into the eighth entry. Then we will have a weight for each of the individual functions as oppose to just weighting unary evidence and pair-wise evidence. Overall, we are free to choose the way to split up the functions. There are many other ways, too. Depends on the task we care about.

4.2 Derivation of multi-class SVM objective

Intuitions for deriving the cost function: We want our score of our ground truth label to be higher than all other possible score that we can get. Think about inference, it means that we are trying to find the element that gives us the highest score. Thus, during learning we want to choose our weights such that our ground truth scores higher than any other possible configuration:

$$\forall \hat{\mathbf{y}} \quad \mathbf{w}^\top f(x^{(i)}, \hat{\mathbf{y}}) \leq \mathbf{w}^\top f(x^{(i)}, \mathbf{y}^{(i)}) \tag{6}$$

How many constraints are there? Size of data set times number of classes. But we dont have to deal with all of the constraints, if it holds for the max $\hat{y}$, then it must hold for the rest of the $\hat{y}$. So the condition becomes:

$$\max_{\hat{\mathbf{y}}} \mathbf{w}^\top f(x^{(i)}, \hat{\mathbf{y}}) \leq \mathbf{w}^\top f(x^{(i)}, \mathbf{y}^{(i)}) \tag{7}$$

4.3 Hinge loss

linearly penalize whenever the maximum score is larger than the ground truth configuration (pay a price). Then we end up with this objective :

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\max_{\hat{\mathbf{y}}} \left(\mathbf{w}^\top f(x^{(i)}, \hat{\mathbf{y}}) + L(\hat{\mathbf{y}}, \mathbf{y}^{(i)}) \right) - \mathbf{w}^\top f(x^{(i)}, \mathbf{y}^{(i)}) \right) \tag{8}$$

As you can see, this formula is very similar to the general formula as shown above, there is no complicated derivation. Notice that one difference between these two formulas is that y is scalar in the general formula and is representing complex object in this formula. This part is called loss-augmented inference. How to explain the penalty term at the end? We want our ground truth to score higher than the maximum of all the configuration. If this is no longer true, that is, if we can find another configuration that scores higher than my ground truth, then we want to pay a price. How much do we want to pay? As much as the difference between the highest score and the score of the ground truth configuration. Importantly, this difference is always non-negative because we can at least plug in y_i to get 0.

And we can use gradient descent to optimize equation (8), define:

$$\min_{\mathbf{w}} L(\mathbf{w}) := \min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\max_{\hat{\mathbf{y}}} \left(\mathbf{w}^\top f(x^{(i)}, \hat{\mathbf{y}}) + L(\hat{\mathbf{y}}, \mathbf{y}^{(i)}) \right) - \mathbf{w}^\top f(x^{(i)}, \mathbf{y}^{(i)}) \right) \tag{9}$$

Iterate:

1. Run loss-augmented inference to find the maximizing argument

2. update the gradient using the argument found in step 1.

And the gradient is:

$$\nabla L(\mathbf{w}) := C\mathbf{w} + \sum_{i\in\mathcal{D}} \big((f(x^{(i)}, \mathbf{y}^*)) - f(x^{(i)}, \mathbf{y}^{(i)})\big) \tag{10}$$

where

$$\mathbf{y}^* = \arg\max_{\hat{\mathbf{y}}} \Big(\mathbf{w}^\top f(x^{(i)}, \hat{\mathbf{y}}) + L(\hat{\mathbf{y}}, \mathbf{y}^{(i)})\Big) \tag{11}$$

Since we are solve one structured prediction task per sample per iteration, equation(9) is very difficult to compute, because $\hat{y}$ is our picture with millions of pixels and exhaustive search doesnt work anymore. However, we can use dynamic programming if the scoring function is a tree. If the inner product is not a tree, we could use the algorithms introduced in previous lectures such as LP relaxation, message passing, and graph cut depending on the application.

5 Structured Learning

A soft version of formula (9) would be:

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{\mathbf{y}}} \exp \frac{L(\mathbf{y^{(i)}}, \hat{\mathbf{y}}) + \mathbf{w}^\top f(x, \hat{\mathbf{y}})}{\epsilon} - \mathbf{w}^\top f(x^{(i)}, \mathbf{y}^{(i)}) \tag{12}$$

Nothing really changes compares to equation (9). This is the most general form (subsumes). The main limitation is that deep learning cant fit in this framework because the score function is linear. The solution to this problem is replace $\mathbf{w}^\top f(x, \mathbf{y})$ with a more general function $F(\mathbf{w}, x, y)$ with $\mathbf{w}$ inside it, where $F(\mathbf{w}, x, y)$ is a score function represented by a computation graph, for example, a deep net.
We can try using gradient descent to optimize the above program. For simplicity, let us ignore the margin term $L(\mathbf{y^{(i)}}, \hat{\mathbf{y}})$ and set $\epsilon = 1$. Let's also define the probability distribution:

$$p(\hat{\mathbf{y}}; x, \mathbf{w}) = \frac{\exp F(\mathbf{w}, x, \hat{\mathbf{y}})}{\sum_{\tilde{\mathbf{y}}} \exp F(\mathbf{w}, x, \tilde{\mathbf{y}})} \tag{13}$$

Then the gradient would be:

$$C\mathbf{w} + \sum_{i\in\mathcal{D}} \big(\sum_{\hat{\mathbf{y}}} p(\hat{\mathbf{y}}; x^{(i)}, \mathbf{w}) \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) - \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \mathbf{y^{(i)}})\big) \tag{14}$$

$$= C\mathbf{w} + \sum_{i\in\mathcal{D}, \hat{\mathbf{y}}} \big(p(\hat{\mathbf{y}}; x^{(i)}, \mathbf{w}) - \delta(\hat{\mathbf{y}} = \mathbf{y^{(i)}})\big) \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) \tag{15}$$

The rearrangement makes sense because when ignoring the first term(Cw), this gradient is zero whenever the prediction is equal to the ground truth. In other words, whenever the predicted probability is identical to the ground truth probability, the gradient is zero. $p^{GT}_{y^{(i)}}(\hat{\mathbf{y}}) = \delta(\hat{\mathbf{y}} = \mathbf{y^{(i)}})$ is called ground truth probability.

The optimization algorithm is as follow:

1. Forward pass to compute $F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}})$

2. Compute $p(\hat{\mathbf{y}}; x, \mathbf{w})$ via soft-max
3. Backward pass via chain rule to obtain gradient
4. Update parameters $\mathbf{w}$

The biggest challenge for this algorithm is that there are too many configurations for the probability distribution. Example of large output space: We can use the trick we used before: assume F is

Figure 3: $|\mathcal{Y}| = 2^{\#tag}$

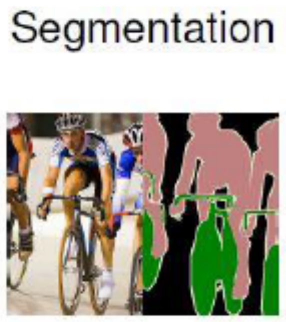

Figure 4: $|\mathcal{Y}| = C^{\#pixels}$

composed of sum of functions that depend on local terms only. Now, instead of having a single CNN, we have a bunch of CNNs:

$$F(\mathbf{w}, x, \mathbf{y}) = F(\mathbf{w}, x, y_1, ..., y_D) = \sum_r f_r(\mathbf{w}, x, \mathbf{y}_r) \tag{16}$$

Compute gradient when F is composed of sum local functions:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} \quad & \textstyle\sum_{i \in \mathcal{D}} \left(\log \sum_{\hat{\mathbf{y}}} \exp F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) - F(\mathbf{w}, x^{(i)}, \mathbf{y}^{(i)}) \right) \\ = & \textstyle\sum_{i \in \mathcal{D}, \hat{\mathbf{y}}} \left(p(\hat{\mathbf{y}}; x^{(i)}, \mathbf{w}) - \delta(\hat{\mathbf{y}} = \mathbf{y}^{(\mathbf{i})}) \right) \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) \\ = & \textstyle\sum_{i \in \mathcal{D}, r, \hat{\mathbf{y}}_r} \left(p_r(\hat{\mathbf{y}}_r; x^{(i)}, \mathbf{w}) - \delta_r(\hat{\mathbf{y}}_r = \mathbf{y}_{\mathbf{r}}^{(\mathbf{i})}) \right) \nabla_{\mathbf{w}} f_r(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}_r) \end{aligned} \tag{17}$$

Then how do we obtain local probability? Generally, there is no way to compute them exactly, but we can obtain approximate marginals $b_r(\hat{\mathbf{y}}_r; x, \mathbf{w})$ via LP relaxation or message passing.

Algorithm: obtain marginals $b_r(\hat{\mathbf{y}}_r; x, \mathbf{w})$ using inference, which is essentially the same thing as solving a loss-augmented inference, then perform gradient steps. For multi-class prediction, we used soft-max.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] A. Schwing. *Pattern Recognition, Lecture 14: Conditional Random Fields, Structured SVMs, Deep Structured Nets*. 2018.