University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 12: Structured Prediction (October 9)

Lecturer: Alexander Schwing Scribe: Harsh Gupta

1 Overview

In this lecture, we will study structured prediction. We will begin with introducing the structured prediction problem and also elaborate on the several applications in which it arises. We will discuss the challenges that make the structured prediction problem different from the basic prediction problem that we have considered so far. We will then focus on techniques and algorithms that allow us to efficiently solve the basic structured inference problem. For additional reading, refer to [1].

2 Recap

2.1 General Framework for Learning

So far, we have established the following generalized framework for learning:

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

In the above framework, we are using a scoring function F for inference. The scoring function is parametrized by $\mathbf{w}$, which is found by minimizing the objective function in (1). L is the taskloss function, which as seen in the case of binary SVM, can take any non-negative value. Recall that the parameter ϵ allows us to switch between different loss functions as suitable for the application, i.e., as $\epsilon \to 0$, we get the hinge-loss and as $\epsilon \to 1$, we get the log-loss.

The above framework is general in the sense that it allows for multiple classes, along with different loss functions depending on the value of ϵ. Moreover, depending on the structure of the function class F, it can allow for arbitrarily complex models to fit the data. Therefore, we can obtain binary/multi-class SVM, binary/multi-class logistic regression as well as deep learning based models, all using the above framework.

2.2 The Inference Problem

As discussed in the previous subsection, the function F is the scoring function that is used for inference, i.e., $F(\mathbf{w},x,\hat{y})$ is the score associated with the data point x being assigned the class $\hat{y}$. The score function is parametrized by $\mathbf{w}$, which is eventually learnt after we solve (1). Naturally, in order to do inference for a data point x, we choose the $\hat{y}$ which maximizes the score function F, i.e.,

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

Now, a natural question that arises is how to solve the optimization problem (2)? As discussed in previous lectures, we can solve an optimization problem over continuous variable spaces using gradient-descent based methods. But, since the variable $\hat{y}$ in (2) lies in a discrete set, we can not use algorithms such as gradient-descent.

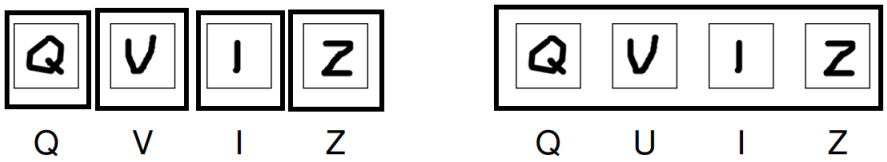

Figure 1: Standard prediction versus structured prediction.

An immediate way to solve (2) is to do an exhaustive search, i.e., compute the function $F(\mathbf{w}, x, \hat{y})$ for all possible values of $\hat{y}$ and then output the $\hat{y}$ (as y^*) that maximizes the function. The advantage of doing exhaustive search is that it is very easy to implement, since it simply requires computing the function F at various points. But, the disadvantage of doing exhaustive search is that for multi-class classification problems with a large number of classes, it can be very slow to compute the function F at all the classes. Hence, it is feasible only for binary classification problems, or classification problems with few classes.

3 Structured Prediction

3.1 Introduction

In order to motivate the utility of structured prediction, let's consider the following example.

Example 1 (Word prediction versus letter prediction) *Consider the prediction problem for the four handwritten letters as shown in Figure 1. When viewed one at a time, as in the standard prediction problem, typical human prediction for the letters is "Q", "V", "I" and "Z". On the other hand, when viewed together as a word, the typical human prediction for the letters is "Q", "U", "I" and "Z", i.e., letters comprising the word "QUIZ". This example shows the power of structured prediction, i.e., making predictions taking correlations across the data points into account, as opposed to standard prediction, where the predictions are made without using the correlations.*

The above example shows the utility of incorporating correlations across data points to make predictions. How do we do that?

Let us consider the above example again. To incorporate correlations in our prediction, we can think of it as a prediction problem over 4 letters at once, as opposed to the standard prediction problem of predicting 1 letter at a time. Therefore, our output y will now lie in the output class $\mathcal{Y} = \{1, 2, 3, ..., 26^4\}$. Clearly, the size of the output class is really large, and hence can be infeasible for inference and learning. To further motivate the utility of structured prediction and better understand how the aforementioned way of handling correlations can lead to disastrously large output class sizes, let's look at a few more examples.

Example 2 (Disparity map estimation) *Consider the problem of disparity map estimation as shown in Figure 2. In this problem, we have two slightly different images from two different sources (Figure 2 shows only one of those images), say the left human eye and the right human eye or two closely spaced cameras. The goal is to estimate the depth of each pixel from the two images. The underlying idea is that deeper objects will move more in the two different images, and objects closer to the cameras will move less. If we solely predict the depth of each pixel by looking at its displacement in the two images, we get the standard independent prediction of depth, as shown in the second image in Figure 2. On the other hand, for each pixel, if we also take into account the displacements of its neighboring pixels, we get better predictions (the third image in Figure 2) as the chances are that pixels in the same neighborhood belong to the same object and have similar depth. This illustrates the power of structured prediction. But, observe that the number of pixels*

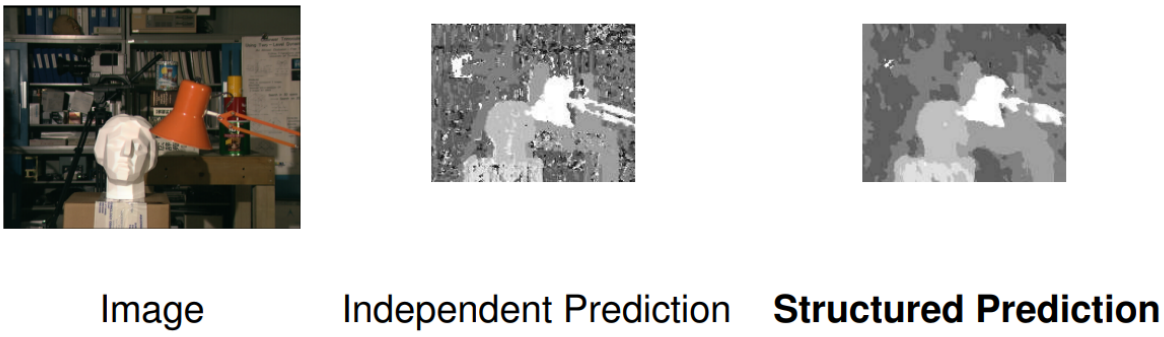

Figure 2: Standard (independent) prediction versus structured prediction in the disparity estimation problem.

Figure 3: Power of structured prediction in the denoising problem.

in the two input images can be very large and hence simply clubbing them all together and treating the problem as a multi-class prediction problem on all pixels will render the problem infeasible due to a very large output space.

Example 3 (Denoising images) *Another important application of structured prediction is denoising, i.e., to smoothen an image which has noisy pixel intensity values. For example, see Figure 3. We have a noisy input image, and our aim is to denoise it and obtain a sharper and smoother version of it. If we consider each pixel independently, there is very little we can do to denoise it, as there is no other information except its own intensity values. On the other hand, for each pixel, if we use the values of its neighboring pixel intensities as well, we can denoise it better, as the chances are that every pixel has similar intensity values as its neighbors. Therefore, structured prediction again proves to be a powerful technique. But, as in the earlier examples, if we simply club all the pixels together and consider the problem to be a multi-class prediction problem, the size of the output space becomes very large, rendering the learning and inference processes infeasible.*

Note: In the standard prediction problem, the output $y \in \mathcal{Y} = \{1, 2, ..., K\}$ (or $\mathcal{R}$) is a scalar output. On the other hand, in structured prediction, given an input $x^{(i)}$, the aim is to output $\mathbf{y}^{(i)} = (y_1^{(i)}, y_2^{(i)}, ..., y_D^{(i)})$, i.e., the output $\mathbf{y}$ is structured and D-dimensional. If, $\forall k$, $y_k \in \mathcal{Y} = \{1, 2, ..., K\}$ (or $\mathcal{R}$), then the output vector $\mathbf{y}$ belongs to $\mathcal{Y}^D$ (or $\mathcal{R}^D$). Therefore, although we can transition between both the formulations as discussed earlier, if the values of K or D are really large, the transitions might lead to infeasible inference and learning problems.

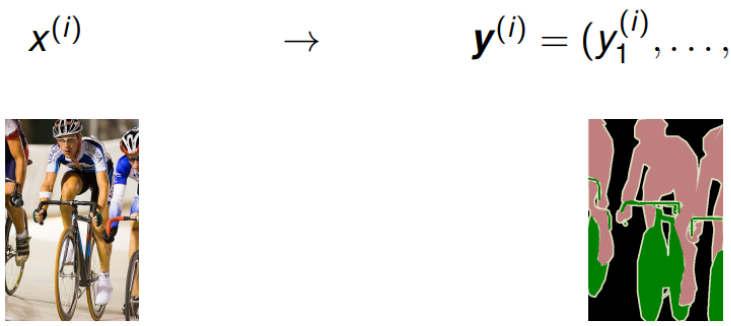

Figure 4: Structured prediction in the image segmentation problem.

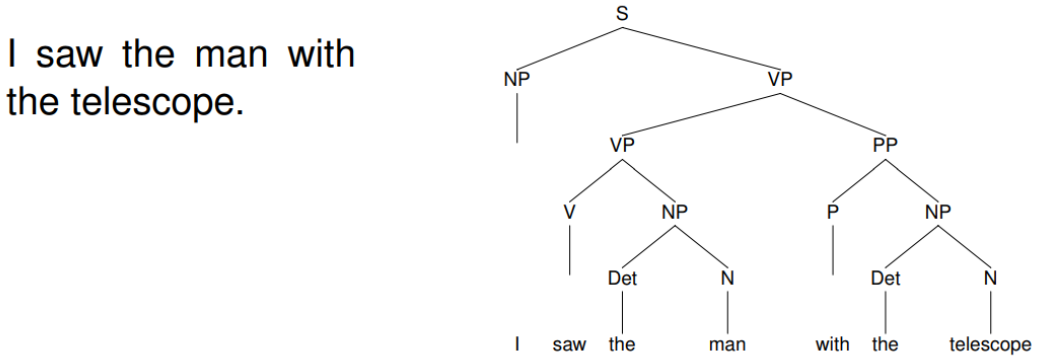

Figure 5: Structured prediction in the sentence parsing problem.

Example 4 (Image segmentation) *In the image segmentation problem, the aim is to label similar parts of the image. For example, in Figure 4, the input image pixels have been classified into three categories: background pixels in black, biker pixels in pink and bike pixels in green. Therefore, for an input image $x^{(i)}$, we have an output $\boldsymbol{y}^{(i)}$ which tells us the category each pixel is classified into. Structured prediction is useful here as the pixel category can be more accurately predicted with the information of its neighboring pixels.*

Example 5 (Sentence parsing) *In the sentence parsing problem, the aim is to estimate a parsing tree for each input sentence, i.e., determining the role each word plays in the sentence. For example, see Figure 5. For each input sentence $x^{(i)}$, we have an output $\boldsymbol{y}^{(i)}$ which tells us the category to which each word in the sentence has been classified into, i.e., whether the word acts as a verb, noun etc. Structured prediction is useful here as the role a word plays in a sentence depends on the entire sentence and not just the word by itself.*

Example 6 (Protein structure estimation) *In the protein structure inference problem, the objective is to infer the folding of a protein along with its secondary and tertiary structure, from its primary structure as an input. For example, see Figure 6. For each primary protein structure input $x^{(i)}$, we have an output $\boldsymbol{y}^{(i)}$ which tells us information about the protein folding and its secondary and tertiary structure. Structured prediction is useful here as the protein folding as well as its*

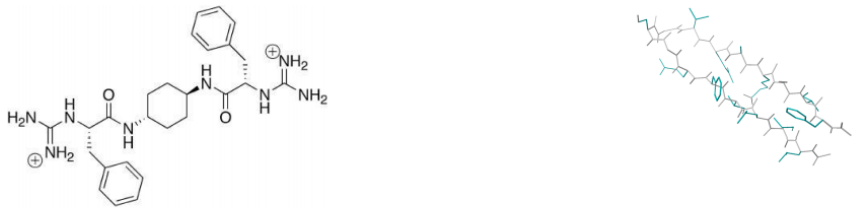

Figure 6: Structured prediction in the protein folding problem.

secondary and tertiary structure depend on its entire amino acid sequence, i.e., the entire primary structure.

3.2 Inference Problem

3.2.1 Introduction

Formally, in structured prediction, for every input x, we have an output $\mathbf{y}$ such that:

$$\mathbf{y} = (y_1, y_2, ..., y_D), \quad y_d \in \{1, 2, ..., K\} \tag{3}$$

Therefore, the inference problem is:

$$\mathbf{y}^* = \arg\max_{\hat{\mathbf{y}}} F(\mathbf{w}, x, \hat{\mathbf{y}}) = \arg\max_{\hat{\mathbf{y}}} F(\mathbf{w}, x, \hat{y}_1, \hat{y}_2, ..., \hat{y}_D). \tag{4}$$

Clearly, if we want to do exhaustive search to solve the above optimization problem, we need to compute and store the score function F at K^D points. For most problems, as we have already observed from the examples discussed before, the values of K and D are large enough to make K^D a very large number, thereby rendering exhaustive search infeasible for solving (4).

3.2.2 Separate Prediction

A natural way to circumvent the problem of large output spaces is to split the objective function in (4) into a sum of D component functions, with each component function being a function of only one component of the output vector $\mathbf{y}$, i.e.,

$$F(\mathbf{w}, x, \hat{y}_1, \hat{y}_2, ..., \hat{y}_D) = \sum_{d=1}^{D} f_d(\mathbf{w}, x, \hat{y}_d) \tag{5}$$

Using the above F as the scoring function in (4), we get:

$$\arg\max_{\hat{\mathbf{y}}} F(\mathbf{w}, x, \hat{y}_1, \hat{y}_2, ..., \hat{y}_D) = \arg\max_{\hat{\mathbf{y}}} \sum_{d=1}^{D} f_d(\mathbf{w}, x, \hat{y}_d) = \sum_{d=1}^{D} \arg\max_{\hat{y}_d} f_d(\mathbf{w}, x, \hat{y}_d) \tag{6}$$

The above inference problem, known as separate prediction, only requires computation and storage of the D component functions on K points each, therefore reducing the computational complexity

to KD as compared to K^D previously. But, the downside of using separate prediction is that it predicts each component of $\mathbf{y}$ separately, thereby not exploiting the correlations among its different components. Although this makes the inference problem computationally feasible, it will lead to poorer results in several examples illustrated before, since exploiting the correlations among the different components of $\mathbf{y}$ is essential to make meaningful predictions in those problems.

3.2.3 Non-trivial Decomposition of the Score Function

We considered the two extreme cases of the structured prediction inference problem in the above subsections. From the computationally infeasible original inference problem exploiting all the correlations across different components in the output, to the separate prediction problem, which completely ignores the correlations across different components in the output and is computationally efficient.

In order to make meaningful predictions, we need to find a way to be computationally efficient while still being able to exploit some correlation. To this end, we can decompose the score function F into less trivial functions as compared to separate prediction, therefore allowing for correlations to play a role in prediction, while simultaneously not being as inefficient as the original inference problem (4). Consider the following decomposition:

$$F(\mathbf{w}, x, \hat{y}_1, \hat{y}_2, ..., \hat{y}_D) = \sum_{r \in \mathcal{R}} f_r(\mathbf{w}, x, \hat{\mathbf{y}}_r) \tag{7}$$

where $r \subseteq \{1, 2, ..., D\}$, are restriction sets and $\mathcal{R}$ is the set of all these restriction sets. For example, if we know that components y_1 and y_2 are correlated, we can impose a restriction set $r = \{1, 2\}$, allowing for a function $f_{\{1,2\}}(\mathbf{w}, x, y_1, y_2)$ in the decomposition of the score function.

For instance, consider Example 1 again. We know that the output is 4-dimensional. From Figure 1, we can observe that the neighboring letters will influence the prediction for a particular letter more than any other letters in the word. Therefore, we can use the following decomposition of the scoring function F:

$$\begin{aligned} F(\mathbf{w}, x, \hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4) = f_1(\mathbf{w}, x, \hat{y}_1) + f_2(\mathbf{w}, x, \hat{y}_2) + f_3(\mathbf{w}, x, \hat{y}_3) + f_4(\mathbf{w}, x, \hat{y}_4) \\ + f_{1,2}(\mathbf{w}, x, \hat{y}_1, \hat{y}_2) + f_{2,3}(\mathbf{w}, x, \hat{y}_2, \hat{y}_3) + f_{3,4}(\mathbf{w}, x, \hat{y}_3, \hat{y}_4) \end{aligned} \tag{8}$$

For the above decomposition, we require a total of 26 computations each for the first 4 functions and a total of 26^2 computations each for the last 3 functions. Therefore, we need a total of $4 \times 26 + 3 \times 26^2$ computations as opposed to 26^4 computations for the inference problem without any decomposition, i.e., Equation (4). Note that the first 4 functions can further be subsumed in at least one of the last 3 functions, reducing the number of computations even further to just 3×26^2.

3.3 Graphical Representation of the Score Function

The score function F can be represented as a graph for ease of analysis and developing efficient inference algorithms. The graph is constructed as follows:

- For each restriction set in the decomposition of the function F, construct a node.
- Construct edges between nodes which have subset relationship between their restriction sets.

For example, corresponding to the score function in Equation (8), the graph construction is shown in Figure 7.

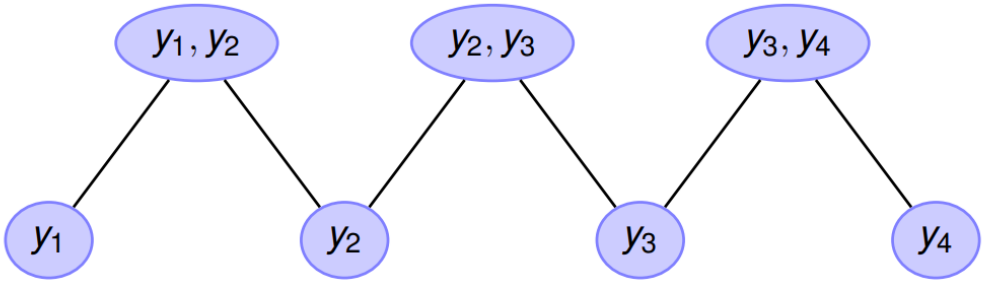

Figure 7: Graph corresponding to the score function F in Equation (8).

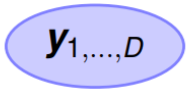

Figure 8: Graph corresponding to the score function F in Equation (4).

For the score functions in Equations (4) and (5), the graph representations are shown in Figures 8 and 9 respectively.

Let's revisit Examples 2 and 4, and obtain decompositions of their score functions, along with the graphical representation of the decompositions. We observed in these examples that the prediction for a particular pixel should intuitively depend on its neighboring pixels as well. Therefore, we can use the following decomposition of the score function in these examples:

$$F(\mathbf{w}, x, \mathbf{y}) = \sum_{d=1}^{D} f_d(\mathbf{w}, x, y_d) + \sum_{i,j} f_{i,j}(\mathbf{w}, x, y_i, y_j) \tag{9}$$

The unary terms in the above decomposition can be encode the effect of the information of the pixel itself on its prediction, i.e., exploiting the local evidence from the image. On the other hand, the binary terms encode the impact of the neighbors on the prediction of a pixel. Therefore, where we expect a pair of neighboring pixels to be similar, we can give higher values to the binary component comprising these pixels for similar prediction. And, where we expect them to be different (for instance when there is a edge in the image segmentation problem), we can assign the binary component lower values for similar prediction. Hence, we can encode a smoothness prior in the

Figure 9: Graph corresponding to the score function F in Equation (5).

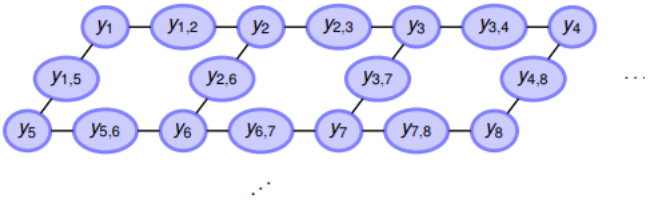

Figure 10: Graph corresponding to the score function F in Equation (9). Also known as a Markov/-conditional random field.

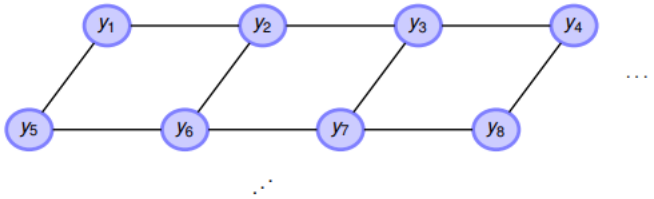

Figure 11: Graph corresponding to the score function F in Equation (9). Also known as a Markov/-conditional random field.

binary components. The graph corresponding to the above score function F is shown in Figure 10.

Note that the graph in Figure 10 can be further simplified by removing the nodes corresponding to $(y_1, y_2), (y_2, y_3)$ and (y_3, y_4) etc., and simply drawing edges between the univariate nodes to denote these restriction sets. This is possible as the univariate restriction sets can be subsumed in one of the multi-variate restriction sets in this case. Figure 11 shows the modified graph. Note that this is a much cleaner graphical representation than before.

3.4 Inference Algorithms

The inference problem on the decomposed score function F (from Equation (7)) is:

$$\mathbf{y}^* = \arg\max_{\hat{\mathbf{y}}} \sum_{r \in \mathcal{R}} f_r(\mathbf{w}, x, \hat{\mathbf{y}}_r) \tag{10}$$

There are several algorithms to solve the above inference problem. Some of these are:

1. Exhaustive search.

2. Dynamic programming.
3. Integer linear program.
4. Linear programming relaxation.
5. Message passing.
6. Graph cut.

In this lecture, we will focus on the first two algorithms and will study the remaining in the next lecture.

3.4.1 Exhaustive Search

As discussed in the previous sections, exhaustive search involves computing the score function $F = \sum_{r \in \mathcal{R}} f_r(\mathbf{w}, x, \hat{\mathbf{y}}_r)$ at all possible values of $\hat{\mathbf{y}}$ and then outputting the value at which the maximum is attained.

The advantage of using exhaustive search is that it is extremely easy to implement. On the other hand, for reasonable sized problems, the computational complexity of exhaustive search can be very bad (since it needs K^D computations), as it requires computing and storing F at each point in the output space.

3.4.2 Dynamic Programming

If the graph representation of the score function in Equation (10) is a tree, we can use dynamic programming to reorganize the terms and efficiently solve the inference problem. Consider the following scoring function F for example:

$$\begin{aligned}\max_{\hat{\mathbf{y}}} F(\mathbf{w}, x, \hat{\mathbf{y}}) = \max_{\hat{\mathbf{y}}} f_3(\mathbf{w}, x, \hat{y}_3) + f_{2,3}(\mathbf{w}, x, \hat{y}_2, \hat{y}_3) + f_2(\mathbf{w}, x, \hat{y}_2) \\ + f_1(\mathbf{w}, x, \hat{y}_1) + f_{1,2}(\mathbf{w}, x, \hat{y}_1, \hat{y}_2)\end{aligned} \tag{11}$$

We can rearrange the terms in the above equation to get:

$$\begin{aligned}\max_{\hat{\mathbf{y}}} F(\mathbf{w}, x, \hat{\mathbf{y}}) &= \max_{\hat{y}_3} \Bigg(f_3(\mathbf{w}, x, \hat{y}_3) + \max_{\hat{y}_2} \big(f_{2,3}(\mathbf{w}, x, \hat{y}_2, \hat{y}_3) + f_2(\mathbf{w}, x, \hat{y}_2) \\ &\qquad + \max_{\hat{y}_1} (f_1(\mathbf{w}, x, \hat{y}_1) + f_{1,2}(\mathbf{w}, x, \hat{y}_1, \hat{y}_2)) \big) \Bigg) \\ &= \max_{\hat{y}_3} \Bigg(f_3(\mathbf{w}, x, \hat{y}_3) + \max_{\hat{y}_2} \big(f_{2,3}(\mathbf{w}, x, \hat{y}_2, \hat{y}_3) + f_2(\mathbf{w}, x, \hat{y}_2) \\ &\qquad + \mu_{1,2 \to 2}(\hat{y}_2) \big) \Bigg)\end{aligned} \tag{12}$$

Therefore, unlike exhaustive search where we will have to do 3^D computations, if we use the above reordered way of solving the inference problem, we only need to do $3D^2$ computations. The graph for the score function F in Equation (11) and the corresponding flow of information according to the Equation (12) is shown in Figure 12.

Generally speaking, dynamic programming has better computational complexity as compared to exhaustive search, whenever it is applicable. For instance, in the case of pairwise models with tree

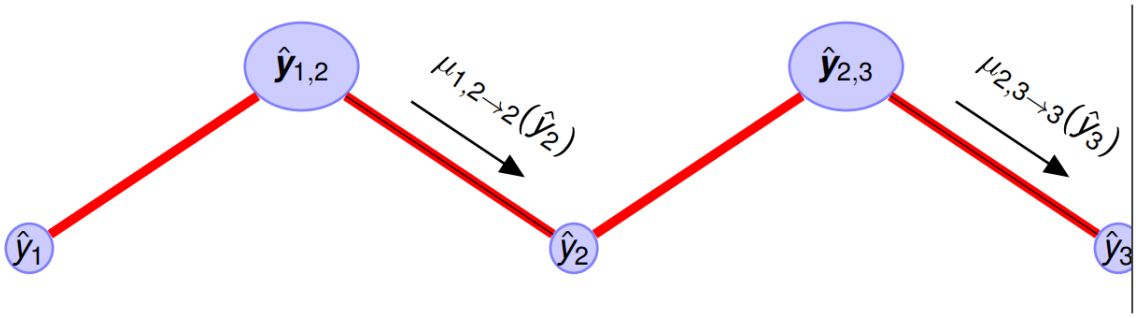

Figure 12: Dynamic programming: graph corresponding to the score function F in Equation (11). Note the arrows depicting the order in which maximization problems are solved as in Equation (12).

representations, the computational complexity of dynamic programming is KD^2, as observed in the previous example. The disadvantage of dynamic programming is that it only works for score functions which have tree representations. For more general loopy graphs, we have to use one of the following methods:

1. Integer linear program.
2. Linear programming relaxation.
3. Dynamic programming extensions (message passing).
4. Graph cut algorithms.

We will cover the above techniques in the next lecture.

4 Quiz Questions

Following were the quiz questions asked at the end of the lecture:

1. Why structured output spaces?
2. What makes computation with structured spaces hard?
3. Inference algorithms for structured output spaces?

Answers:

1. To exploit correlation among different components of the output, which might arise naturally in the problem.
2. Using structured output spaces leads to large output spaces, thereby rendering standard prediction and inference methods computationally infeasible.
3. Depending on the structure of the problem and computational resources available, one can use different inference algorithm for structured output spaces. For example, exhaustive search and dynamic programming.

References

[1] D. Koller, N. Friedman, and F. Bach. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.