University of Illinois Fall 2018

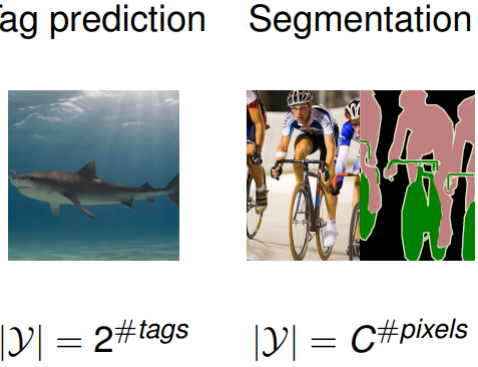
ECE 544NA: Pattern Recognition
Lecture 14: October 16

Lecturer: Alexander Schwing Scribe: Sohrab Madani

1 Introduction

In the last two lectures (i.e. lectures 12 and 13), we covered structured prediction and we saw that the hardness of structured prediction boils down to the very large size of the output space (figure 1). To address this, we discussed various structured inference algorithms, however, we did not go over how we can learn in structured contexts, and how should our learning framework be altered to incorporate structured output spaces. These are the questions that we will address in the present lecture.

Figure 1: Structured objects have a large output space ($\mathcal{Y}$)



Note 1. In the following, by *previous lectures*, we mean lectures 12 and 13, unless otherwise specified.

Note 2. Structured output is denoted by boldface y ($\boldsymbol{y}$). Specifically, $\boldsymbol{y} = (y_1, \ldots, y_D)^\top$ where each y_i is an element of $\{0, \ldots, K-1\}$. For instance, y could be an image and each y_i a pixel, having an intensity that ranges from 0 to 255.

2 Recap

So far, this is our general framework for learning:

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

where F is our scoring function and L is taskloss. Also keep in mind that with this framework we can cover logistic regression, binary and multiclass SVM, multiclass regression, and deep learning using our degrees of freedom (i.e. playing around with ϵ, F, L, and domain of y).

Remember that $\boldsymbol{y}^{(i)}$ (as opposed to $y^{(i)}$), represents the structured object discussed in previous lectures. We saw that due to their large size, it is sometimes not feasible to find the configuration

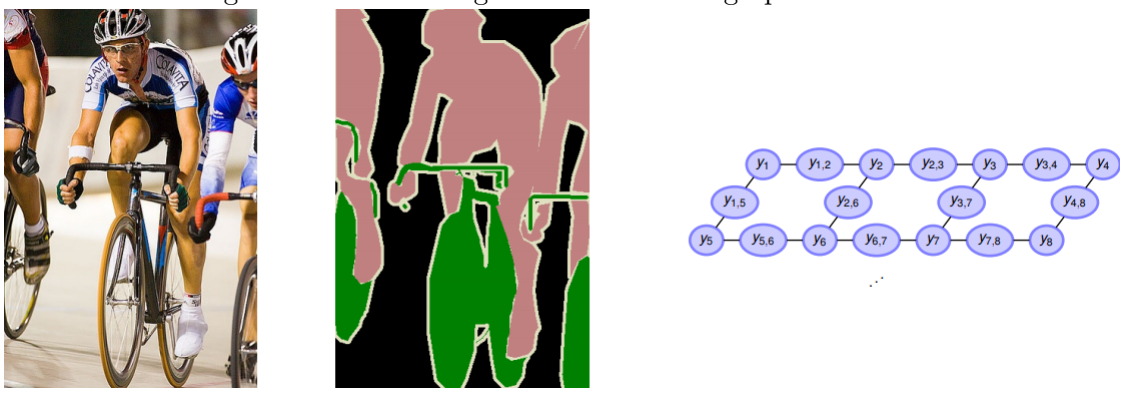
that maximizes the scoring function F (i.e. inference) by doing exhaustive search. This was solved by assuming that F can be decomposed into a summation of local functions, each depending on a few y's denoted by $\boldsymbol{y}_r$. This allowed us to come up with more efficient algorithms for inference[1]. The main question in this lecture is, how can we incorporate $\boldsymbol{y}$ into our learning formulation? Can we simply replace $y^{(i)}$ with $\boldsymbol{y}^{(i)}$ in equation (1)?

3 Feature Vector for Structured Objects

3.1 An Example

Let us start this section by looking at the problem of semantic segmentation (i.e. associating a class label to each pixel of an image), as we will refer to this problem several times in the remaining of this lecture. In previous lectures, we considered a grid-like graphical model for this problem. We saw that *unary potentials* give us local evidence, while *pairwise potentials*, encourage smoothness.

Figure 2: Semantic segmentation and its graphical model



Let us first clarify the meaning of *encouraging smoothness* through an example.

Example 1. For the sake of simplicity, consider a simple case in which f_{12} is a pairwise term that depends on y_1 and y_2, and each y_i can have a value of 0 or 1. f_{12} can be represented by a 2-by-2 matrix, with row 1 corresponding to $y_1 = 0$, row 2 corresponding to $y_1 = 1$ and columns corresponding to y_2 in a similar manner. Then we can put

$$f_{12} = \alpha \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2}$$

where α is an arbitrary positive constant. Note that

$$f_{12} = \begin{cases} \alpha & \text{if } y_1 = y_2 \\ 0 & \text{if } y_1 \neq y_2 \end{cases} \tag{3}$$

which means that we will get a higher score if $y_1 = y_2$. In other words, we are *encouraging* the case where y_1 and y_2 are equal.

[1]Of course, not all algorithms yield exact solutions, and not all can be applied to all problems. In other words, accuracy and efficiency of inference algorithms is problem dependent.

3.2 Feature Vector

Suppose that we have trained a classifier for our problem that gives us local evidence. An immediate question now is, given the unary potential classifier and the smoothness prior, how can one combine the two to get the best possible result (e.g. what value should be used for α in Example 1)? In other words, how do we trade unary potentials with pairwise potentials? We will try to formulate this using the tools we have developed thus far. To this end, let us consider the case in which the scoring function $F()$ is of the form[2]

$$F(\mathbf{w}, x^{(i)}, \boldsymbol{y}^{(i)}) = \mathbf{w}^\top f(x^{(i)}, \boldsymbol{y}^{(i)}). \tag{4}$$

Assume that the output of the function f is a vector with M elements, $f = (f_1, f_2 \ldots, f_M)^\top$ and each of f_m $(1 \leq m \leq M)$, can be decomposed into a summation of functions $f_{m,r}$ just like in previous lectures:

$$f(x^{(i)}, \boldsymbol{y}^{(i)}) = \begin{bmatrix} f_1(x^{(i)}, \boldsymbol{y}^{(i)}) \\ f_2(x^{(i)}, \boldsymbol{y}^{(i)}) \\ \vdots \\ f_M(x^{(i)}, \boldsymbol{y}^{(i)}) \end{bmatrix} = \begin{bmatrix} \sum_r f_{1,r}(x^{(i)}, \boldsymbol{y}_r^{(i)}) \\ \sum_r f_{2,r}(x^{(i)}, \boldsymbol{y}_r^{(i)}) \\ \vdots \\ \sum_r f_{M,r}(x^{(i)}, \boldsymbol{y}_r^{(i)}) \end{bmatrix}. \tag{5}$$

Combining this with equation (4) we get

$$F(\mathbf{w}, x^{(i)}, \boldsymbol{y}^{(i)}) = \mathbf{w}^\top f(x^{(i)}, \boldsymbol{y}^{(i)}) = \sum_{m=1}^{M} \mathrm{w}_m \underbrace{\Big(\sum_r f_{m,r}(x^{(i)}, \boldsymbol{y}_r^{(i)}) \Big)}_{\text{individual models}} = \underbrace{\sum_r \hat{f}_r(\mathbf{w}, x^{(i)}, \boldsymbol{y}_r)}_{\text{combined model}}. \tag{6}$$

Here, $f()$ is our feature vector, and contains as its elements M independent models. Each f_m can be treated as a function that corresponds to a graphical model of its own. Depending on the problem we are trying to solve, we get to choose M and f_m $(1 \leq m \leq M)$. Let us take a look at some examples.

Example 2. In the case of semantic segmentation, we can take $M = 2$, and take f_1 to be a summation of unary potentials, and f_2 be a summation of pairwise potentials that encourage smoothness. The scoring function will then be $\mathbf{w}_1 f_1 + \mathbf{w}_2 f_2$. Now the problem of how to trade between unary and pairwise potentials can be solved if we can somehow *learn* the vector $\mathbf{w}$, which is discussed in the next section.

Example 3. In Example 1, say we do not necessarily want to have f_{12} to be of the form of equation (2). Instead, we want our model to *learn* each element of the matrix that represents f_{12}. Then the solution would be to have four different functions $f_{12}^{(1)}, \ldots, f_{12}^{(4)}$ in different elements of the feature vector f, such that

$$f_{12}^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad f_{12}^{(2)} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad f_{12}^{(3)} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad f_{12}^{(4)} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}. \tag{7}$$

When these four get combined in equation (6), the model will have the matrix for the pair y_1 and y_2 as

$$\begin{bmatrix} \mathrm{w}_{m_1} & \mathrm{w}_{m_2} \\ \mathrm{w}_{m_3} & \mathrm{w}_{m_4} \end{bmatrix}$$

for some $m_1, \ldots, m_4 \in \{1, \ldots, M\}$, where the weights are to be learned.

[2]This means that we are considering the above $F()$ for equation (1), which gives us the equation for linear multi-class formulation.

4 Structured SVM

In order to arrive at a formulation, we will first take a quick look at how we initially derived the multi-class SVM objective.

4.1 Review of Multi-Class SVM

In lecture 9, multi-class SVM, we wanted to find $\mathbf{w}$ such that the score for ground truth is greater than (or equal to) every other configuration. We then linearly penalized whenever the maximum configuration score was within a margin $L(y^{(i)}, \hat{y})$ of the score for ground truth. By choosing $L(y^{(i)}, \hat{y})$ to be 1, we derived the hinge-loss formulation:

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\max_{\hat{y}} \left(1 + \mathbf{w}^\top \psi(x^{(i)}, \hat{y})\right) - \mathbf{w}^\top \psi(x^{(i)}, y^{(i)}) \right). \tag{8}$$

Finally, we extended our model by replacing the hinge-loss function with a more general framework to get

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + \mathbf{w}^\top \psi(x^{(i)}, \hat{y})}{\epsilon} - w^\top \psi(x^{(i)}, y^{(i)}) \right). \tag{9}$$

Remember that we refered $L(y^{(i)}, \hat{y}) + \mathbf{w}^\top \psi(x^{(i)}, \hat{y})$ as loss-augmented inference.

4.2 structured learning framework for SVM

Let us follow the above steps to derive a similar learning framework for structured output space data, where y is boldface ($\boldsymbol{y}$). First, we want to find $\mathbf{w}$ such that

$$\forall \hat{\boldsymbol{y}} \qquad \mathbf{w}^\top f(x^{(i)}, \hat{\boldsymbol{y}}) \leq \mathbf{w}^\top f(x^{(i)}, \boldsymbol{y}^{(i)}). \tag{10}$$

Note that checking this for every possible $\hat{\boldsymbol{y}}$ requires exponential time, as covered in previous lectures. An equivalent to equation (7) would be

$$\max_{\hat{\boldsymbol{y}}} \left(\mathbf{w}^\top f(x^{(i)}, \hat{\boldsymbol{y}})\right) \leq \mathbf{w}^\top f(x^{(i)}, \boldsymbol{y}^{(i)}). \tag{11}$$

It is possible now to define a linearly penalizing loss function, which gives rise to the learning framework

$$\min_{\mathbf{w}} L(\mathbf{w}) := \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\max_{\hat{\boldsymbol{y}}} \left(L(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}) + \mathbf{w}^\top f(x^{(i)}, \hat{\boldsymbol{y}})\right) - \mathbf{w}^\top f(x^{(i)}, \boldsymbol{y}^{(i)}) \right). \tag{12}$$

Equation (12) is often referred to as Structured Support Vector Machine [**?**], or Max-Margin Markov Network [**?**]. Note that equation (12) is the same as equation (9), except that y has become boldface, ψ is replaced by f, and a general $L()$ is assumed.

Finding the maximum of $\mathbf{w}^\top f(x^{(i)}, \hat{\boldsymbol{y}})$ is what we called inference in previous lectures, and the argument inside max in equation (12) is the same, except that it has the loss $L()$ added to it, which is why this part is called loss-augmented inference.

Algorithm 1 Gradient descent for structured SVM

while *stopping criteria not reached* **do**

1. (Loss-augmented inference) solve $\arg\max_{\hat{\boldsymbol{y}}} \left(L(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}) + \mathbf{w}^\top f(x^{(i)}, \hat{\boldsymbol{y}})\right)$
2. Perform step gradient: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w})$

end

Equation (12) can be optimized using conventional methods, such as gradient descent. At each step of the gradient descent, loss-augmented inference should be maximized (see Algorithm 1).

To see how complicated this is, observe that we will have to solve one structured prediction (i.e. solve the structured loss-augmented inference) per sample (i.e. for all $i \in \mathcal{D}$) per iteration (i.e. for each time $\mathbf{w}$ gets updated). In the next section, we will generalize the framework we developed in this section.

5 Structured Learning

Just like in the last step of review in section 4.1, equation (12) for structured SVM can be generalized to a more general framework:

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\epsilon \ln \sum_{\hat{\boldsymbol{y}}} \exp \frac{L(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}) + \mathbf{w}^\top f(x^{(i)}, \hat{\boldsymbol{y}})}{\epsilon} - \mathbf{w}^\top f(x^{(i)}, \boldsymbol{y}^{(i)}) \right). \tag{13}$$

Similar to lecture (9), this framework subsumes different problems:

- Linear structured SVM
- Linear conditional random fields (for more information, see [**?**])
- binary/multi-class SVM and logistic/multi-class regression

If one were to use the framework in equation (13), say, for semantic segmentation described in section 3, two steps of learning would be required. The first step would be to run a classifier to obtain local evidence, and the second step would then be to use the scores from this pre-trained classifier to learn how to combine local evidence with, for instance, smoothness. One interesting question would be ask if it is possible to avoid this two-step learning process, and try to learn the classifier for unary potentials jointly with the combination part.

Another motivation to change the current framework comes from the observation that there is an inner product between $\mathbf{w}$ and $f()$ in equation (13), which calls for a generalization to (13) in order to overcome this limitation.

The same idea that we saw in previous lectures also applies here. The idea is to replace the inner product between $\mathbf{w}$ and $f()$ with a more general function, in hopes of being able to learn them at the same time, in which case our framework would be

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\epsilon \ln \sum_{\hat{\boldsymbol{y}}} \exp \frac{L(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}) + F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}})}{\epsilon} - F(\mathbf{w}, x^{(i)}, \boldsymbol{y}^{(i)}) \right). \tag{14}$$

One could think of $F()$ as a score function represented by a computation graph, for example a deep net. To optimize this program, let us proceed with gradient descent. Say we want to solve

the optimization problem in equation (14) using gradient descent. For simplicity while computing the gradient, we will assume $\epsilon = 0$ and $L = 0$. Define

$$p(\hat{\boldsymbol{y}}; x, \mathbf{w}) = \frac{\exp\left(F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}})\right)}{\sum_{\widetilde{y}} F(\mathbf{w}, x^{(i)}, \widetilde{\boldsymbol{y}})}$$

Now the gradient of equation (14) with respect to $\mathbf{w}$ would be

$$C\mathbf{w} + \sum_{i \in D} \left(\sum_{\hat{\boldsymbol{y}}} p(\hat{\boldsymbol{y}};\, x^{(i)}, \mathbf{w})\, \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}}) - \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \boldsymbol{y}^{(i)}) \right), \tag{15}$$

which could be further simplified to

$$C\mathbf{w} + \sum_{i \in \mathcal{D}, \hat{\boldsymbol{y}}} \Big(p(\hat{\boldsymbol{y}};\, x^{(i)}, \mathbf{w}) - \underbrace{\delta(\hat{\boldsymbol{y}} = \boldsymbol{y}^{(i)})}_{p^{\mathrm{GT}}_{\boldsymbol{y}^{(i)}}(\hat{\boldsymbol{y}})} \Big)\, \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}}) \tag{16}$$

where the notation from the underbrace comes from when we introduced cross-entropy loss in lecture (10). It is also noteworthy that even though we are dealing with structured objects, we will get a zero gradient (neglecting the regularization term at the beginning) if we can predict the ground truth. Algorithm (2) summarizes the optimization algorithm for equation (14).

Algorithm 2 Gradient descent for optimizing equation (14)

while *stopping criteria not reached* **do**

1. Forward pass to compute $F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}})$
2. Compute $p(\hat{\boldsymbol{y}}; x, \mathbf{w})$ via soft-max
3. Backward pass via chain rule to compute gradient
4. Perform gradient step: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w})$

end

The main challenge in Algorithm (2) is the first step. If the data is structured, the function F will have exponential number of scores, which makes Algorithm (2) computationally not feasible. We have to find a better way to represent F if the output space is very large. After all, equation (14) is essentially the same as what we had derived earlier for deep nets (lectures 10 and 11), except that here y has become boldface.

The idea here is once again one that we have already seen in this and previous lectures, and that is to assume that F can be decomposed into a summation of local f_r's, or more formally,

$$F(\mathbf{w}, x, \boldsymbol{y})) = F(\mathbf{w}, x, y_1, \ldots, y_D) = \sum_r f_r(\mathbf{w}, x, \boldsymbol{y}_r), \tag{17}$$

where r changes over subsets of $\{1, \ldots, D\}$. Now every local function f_r is an arbitrary composite function (i.e. we do not impose any further limitations of f_r). For instance, f_r's can be CNNs.

The gradient of equation (14), with the simplifications made earlier (plus neglecting the regularization term), will be

$$\begin{aligned} &\frac{\partial}{\partial \mathbf{w}} \sum_{i \in \mathcal{D}} \left(\log \sum_{\hat{\boldsymbol{y}}} \exp\left(F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}})\right) - F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}}) \right) \\ &= \sum_{i \in \mathcal{D}, \hat{\boldsymbol{y}}} \Big(p(\hat{\boldsymbol{y}};\, x^{(i)}, \mathbf{w}) - \delta(\hat{\boldsymbol{y}} = \boldsymbol{y}^{(i)}) \Big)\, \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}}) \\ &= \sum_{i \in \mathcal{D}, r, \hat{\boldsymbol{y}}_r} \Big(p_r(\hat{\boldsymbol{y}}_r;\, x^{(i)}, \mathbf{w}) - \delta_r(\hat{\boldsymbol{y}}_r = \boldsymbol{y}_r^{(i)}) \Big)\, \nabla_{\mathbf{w}} f_r(\mathbf{w}, x^{(i)}, \hat{\boldsymbol{y}}_r). \end{aligned} \tag{18}$$

The obvious question now is how to obtain the marginals $p_r(\hat{\boldsymbol{y}}_r; x^{(i)}, \mathbf{w})$. It turns out that computing these marginals is a hard and computationally expensive problem. There are ways to obtain approximations of $p_r(\hat{\boldsymbol{y}}_r; x^{(i)}, \mathbf{w})$'s, such as *beliefs* denoted by $b_r(\hat{\boldsymbol{y}}_r; x^{(i)}, \mathbf{w})$, using methods such as LP relaxation and Message Passing. Note that for each iteration over $\mathbf{w}$, we would have to compute these approximation for marginals.

Before we go furgher, take a look at Algorithm 3, which summarizes the high-level general learning algorithm for any problem.

Algorithm 3 General algorithm for learning

while *stopping criteria not reached* **do**
  1. **while** *stopping criteria not reached* **do**
    update marginals b_r
  end
  2. compute gradient perform gradient step: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w})$ (or more generally, update parameters $\mathbf{w}$)
end

Algorithm 3 on a high level, subsumes all the learning algorithms we have discussed thus far, Even for the simple case of binary classification. The difference is that in binary classification, we can compute the marginals much more easily using soft-max.

As previously explained, in algorithm 3, the expensive part of computation is again the first step. This suggests that the answer to the earlier question of whether or not it is possible to work our way around the two-step learning in structured output space is indeed negative (at least, as far as we are concerned in this course). Algorithm (4) describes the algorithm for learning in structured objects spaces using equation (18).

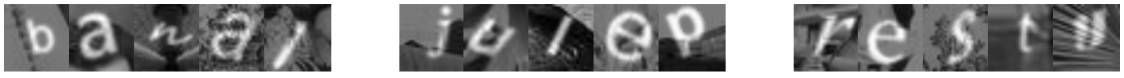
Algorithm 4 deep structured learning

while *stopping criteria not reached* **do**
  1. Forward passes to compute $f_r(\mathbf{w}, x, \hat{\boldsymbol{y}}_r)$ for all r
  2. Estimate beliefs $b_r(\hat{\boldsymbol{y}}_r, x, \mathbf{w})$, exactly or approximately
  3. Compute the difference between estimated and groundtruth beliefs
  4. Back-propagate differences to obtain gradient $\nabla_{\mathbf{w}} L(\mathbf{w})$
  5. Update parameters $\mathbf{w}$ by performing the gradient step: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w})$
end

6 Examples

Example 4. The problem of finding words from distorted images of letters was used in previous lectures to motivate the structured output space. Let us now look at the results from an example of 5-letter word detection within distorted images. The input images are in the form of figure 3.

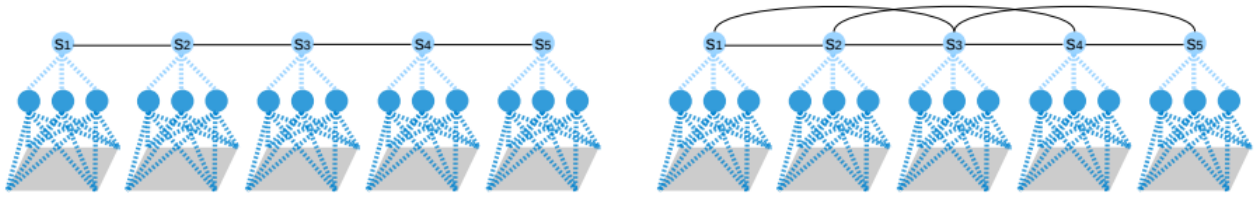
Figure 3: Input data for Example 4



Three different learning solutions are implemented to solve the problem:

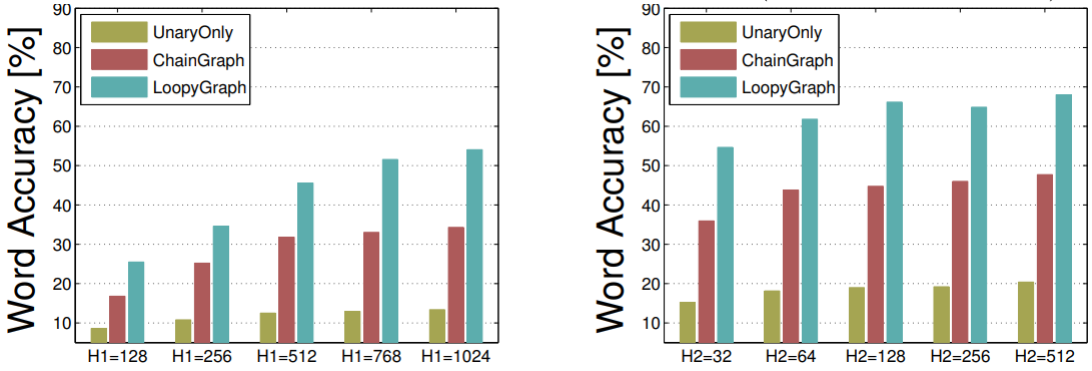
1. Using only unary functions, which does not consider the correlation between different letters in a word. This case is easy to solve.
2. Using a first-order Markov chain which considers the correlation between consecutive letters (this result in a chain graph, look at figure 4). This case also can be solved exactly through dynamic programming described in previous lectures, since the graph we are dealing with is a tree.
3. Using a second-order Markov chain (loopy graph) which considers the correlation between consecutive letters, and also letters that are two hops away (this results in a loopy graph, look at figure 4).

Figure 4: Firstorder Markov chain (left) and second-order Markov chain



Each of the tree solutions were implemented in two different configurations of one-layer and two-layer MLP[3]. The results of these three implementations are demonstrated in figure 5. It can be seen that word accuracy increases as the MLP layers increase, and the learning gets more structured. The general conclusion is that as learning gets deeper and more structured, the performance will get better.

Figure 5: Results for different solutions and MLP layers (one-layer MLP on left)



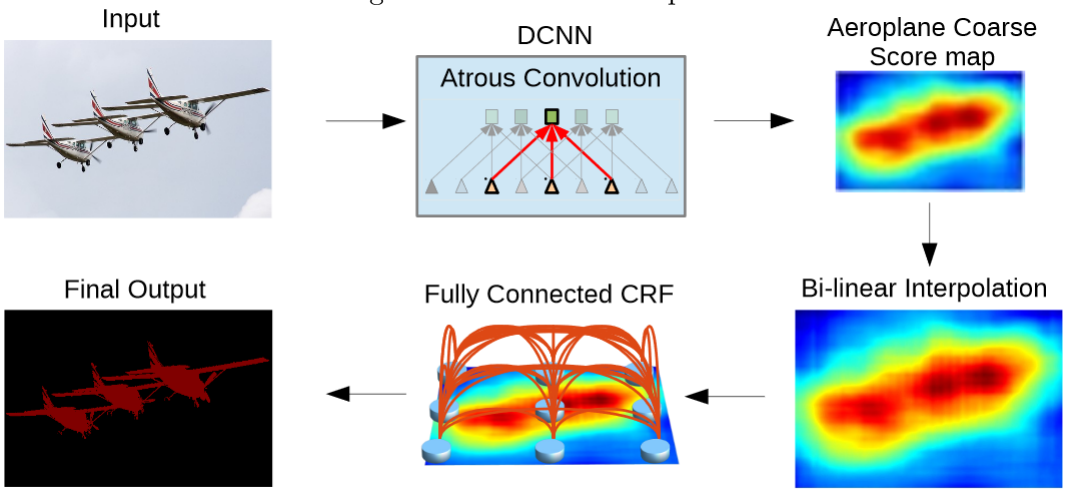
For more information about these results, refer to [**?**].

Example 5. We will look at example of semantic image segmentation. More information this examples can be found in [**?**].

Here, we will give a summary of the model. The input first goes through a deep CNN. Then a Bi-linear interpolation is used to enlarge the feature maps. Finally, a fully-connected conditional random field (or CRF, see section 5) is applied to refine the segmentation. Look at figure 6 for an illustration of the model used in this example.

[3]MLP stands for Multi-Layer Perceptron. For more information, see http://deep learning.net/tutorial/mlp.html

Figure 6: model for example 5



The part with CRF has been employed to provide structured prediction for finer boundary detection. More specifically, there is a trade-off between localization accuracy and classification performance in deep CNNs. Put differently, deep CNNs can successfully predict the rough position of the object, but cannot delineate its borders very well, and this is where the CRF comes in.

Some of the results from [**?**] are shown in figure 7. As you can see, before CRF, objects are detected but have coarse boundaries, whereas after CRF, boundaries are finer and more aligned with ground truth.

Figure 7: results for example 5. before and after refer to the CRF block in figure 6.