University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 14: October 16

Lecturer: Alexander Schwing Scribe: Ziyang Liu

1 Review

1.1 General learning framework

First we are going to review the general framework for our learning program.

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

$F(w, x^{(i)}, y^{(i)})$ is the score of ground truth configuration. ϵ is some positive constant. $L(y^{(i)}, \hat{y})$ is the taskloss which indicates the difference between the prediction configuration and ground truth configuration.

Base on these general framework, we can get other machine learning models.

To get logistic regression, we set $\epsilon = 1$.

To get SVM, we set $\epsilon = 0$.

To get deep nets, the score function has nested structure like this.

$$F(\mathbf{w}, x, y) = f_1(\mathbf{w_1}, y, f_2(\mathbf{w_2}, f_3(...f_n(\mathbf{w_n}, x))))$$

1.2 Inference

Inference is about how to find the highest scoring configuration. For example, in binary classification case, we just have a vector and multiply it with weights vector and if the result is postive, we classify it into one class otherwise, we say it belongs to the other class. For multi-class cases, we basically are multiplying with feature vector and pick the highest score among all classes.

We can use structured prediction to model the correlation between samples. However, sometimes the searching space are very large, for example, if we want to predict each pixel to see whether it is background or human or bicycle. If the image has 1000 pixels, the search space are up to 3^{1000} which is too large for exhaustive search. Therefore, we want to decompose the $F()$,the scoring function, into some structures and use other efficient algorithms to perform the inference. We introduced other inference algorithms like Dynamic programming, integer linear program, linear programming relaxation, message passing, and graph-cut. But each inference algorithm has their own applied senario, for example, dynamic programming can only be used in tree shaped structure.

2 Learning in Structured Prediction

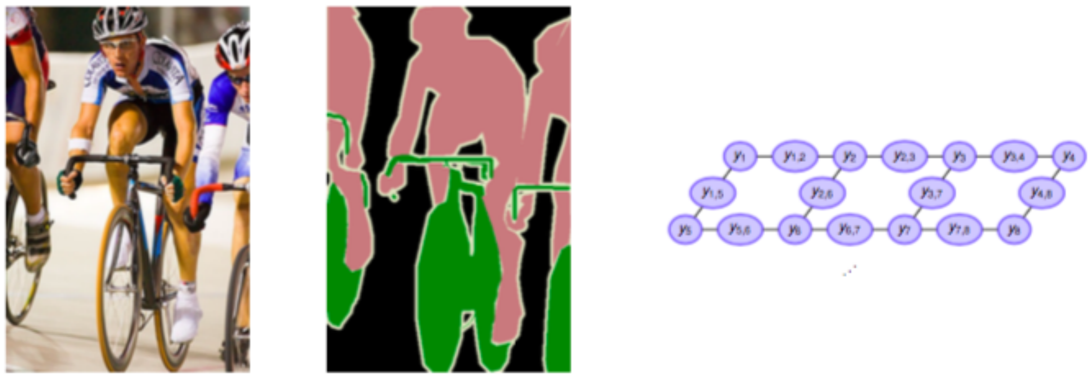


Figure 1: Semantic Segmentation

First we are going to give an example about complex object inference $\mathbf{y}^{(i)}$, as opposed to $y^{(i)}$.

In Figure 1, our input is an image and we want our inference model to be able to generate an output image that classify each pixel in original image to one of three classes(background, human, bicycle). In the right side of Figure 1, we see there are local evidence like $y_1, y_2, ..., y_8$, we also call them unary potentials. We also have $y_{1,2}, ..., y_{7,8}$ those pairwise potentials. One concept which is introduced by pairwisse potentials is called smoothness. Basically it means we want to classify neighboring pixels into the same class. But the question is how much weight we should assign to local evidence as against to pairwise evidence. In another word, how much correlations we want to utilize? We cannot manually tune the weights since that is just unpractical. Therefore, we should seek the way to put this complex $\mathbf{y}^{(i)}$ into our general learning framework and let the machine learn it by itself.

We can start with linear multi-class formulation, our general framework looks like the following.

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

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

Then what we want to do right now is to put our $\mathbf{y}$ into our framework, which means we need to figure out what $F(\mathbf{w}, x^{(i)}, \mathbf{y})$ is. It is what we called feature vector and we are going to explain it in details in the next section.

2.1 Feature Vector

Continuing with our discussion about the semantic segmentation task, we see that $y_1, y_2, ...$ are local function and we build $y_{1,2}, ...$ as pairwise function which create the relation between pixels.

Local function can give us local evidence that could come from some deep nets. It just tells what the deep net think the pixel is background, human, or bicycle. Pairwise function is used to enforce some correlations like pixels that next to each other should normally have same labels. Figure 2 shows the feature vector that composed by local functions and pairwise functions.

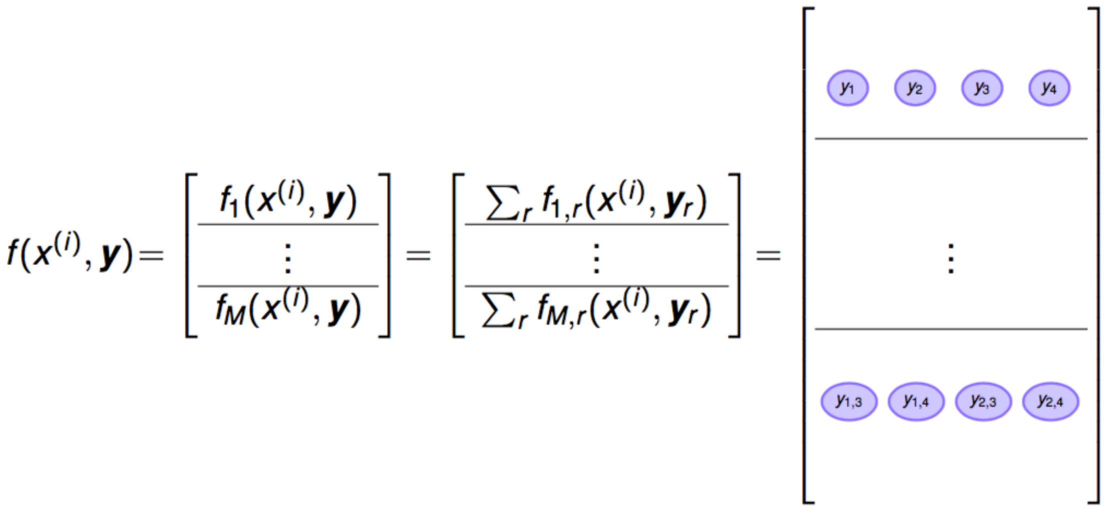

$f(\boldsymbol{x}^{(i)}, \boldsymbol{y}) = \begin{bmatrix} f_1(\boldsymbol{x}^{(i)}, \boldsymbol{y}) \\ \vdots \\ f_M(\boldsymbol{x}^{(i)}, \boldsymbol{y}) \end{bmatrix} = \begin{bmatrix} \sum_r f_{1,r}(\boldsymbol{x}^{(i)}, \boldsymbol{y}_r) \\ \vdots \\ \sum_r f_{M,r}(\boldsymbol{x}^{(i)}, \boldsymbol{y}_r) \end{bmatrix} =$
y_1
y_2
y_3
y_4
$\vdots$
$y_{1,3}$
$y_{1,4}$
$y_{2,3}$
$y_{2,4}$

Figure 2: Feature vector

The notation in the Figure 2 is a little bit confusing. From my understanding, we can just ignore the notation first and keep the concept that feature vector just stacks up multiple graphical models in mind. Then we look back at Figure 2. On the right side of the figure, we see that on the top of the vector, there are separated local functions. So this could be a graphical model basically tells that our program would take each pixel individually into account. But sometimes we may think pixel1 and pixel3 might have some relations. Pixel1 and pixel4 might also have some relations. Therefore, we come up with another graphical model which is just like the bottom of the vector in Figure 2. We can also think pixel at the top left corner might have some relations with pixel at bottom right corner and come up with some graphical model. But that is normally a bad idea since intuitively, pixel at top left corner should not have much impacts on the pixel at bottom right corner. Therefore, the graphical model is totally decided by the developers. Good graphical models could improve the performance our program. Then we look back at the notation, M, r are just a way to express the relations arrangement that developers defined arbitrarily.

At this moment, we can express the scoring function as the weight matrix multiplied by the feature vector and it should look like the following.

$$F(\mathbf{w}, x^{(i)}, \mathbf{y}) = \mathbf{w}^T f(x^{(i)}, \mathbf{y})$$

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

Next, we should start to care about how to learn the weigths. We would use multi-class SVM as the example to demonstrate.

2.2 Learning with SVM

When we are dealing with multi-class cases, we should have the intuition that we want the score of our ground truth label to be the highest.

$$\forall \hat{\mathbf{y}}, \mathbf{w}^T f(x^{(i)}, \hat{\mathbf{y}}) \leq \mathbf{w}^T f(x^{(i)}, \hat{\mathbf{y}})$$

The right side of the inequality above is the ground truth's score.

In another word we can express it as

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

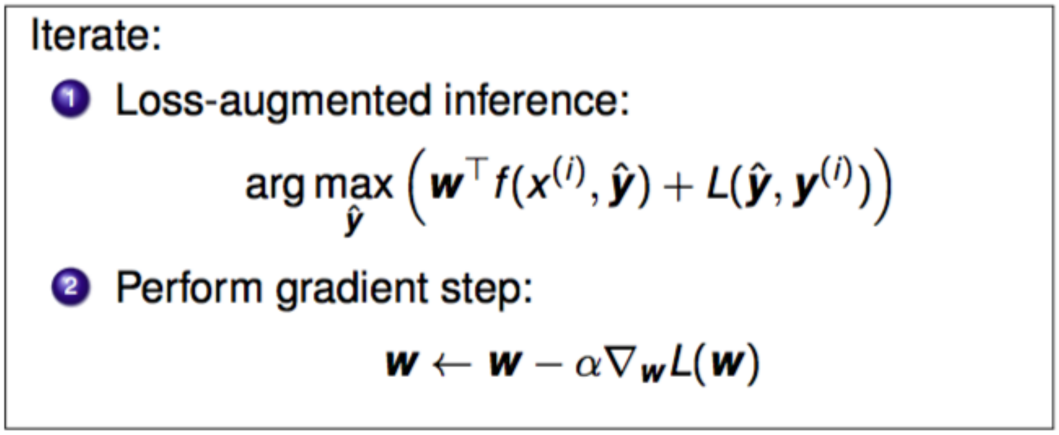
Whenever the maximum of our taskloss plus score from prediction is greater then our ground truth score, we need to penalize it. In another word, whenever we find a configuration whose score is higher than the ground truth, we want to pay a cost. Therefore, objective function becomes following

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

We call $\max_{\hat{\mathbf{y}}}(\mathbf{w}^T f(x^{(i)}, \hat{\mathbf{y}}) + L(\hat{\mathbf{y}}, \mathbf{y}^{(i)}))$ loss-augmented inference.

Next, we need to optimize our objective function.

2.3 Optimize Structured SVM



Iterate:

1. Loss-augmented inference:

$$\arg\max_{\hat{\boldsymbol{y}}} \left(\boldsymbol{w}^\top f(x^{(i)}, \hat{\boldsymbol{y}}) + L(\hat{\boldsymbol{y}}, \boldsymbol{y}^{(i)})\right)$$

2. Perform gradient step:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} L(\boldsymbol{w})$$

Figure 3: Gradient descent algorithm

We can use gradient descent to optimize our program. The algorithm is described in Figure 3. Basically, it just says we are going to iterate our program. For each iteration, we firstly are going to find the prediction $\hat{\mathbf{y}}$ that makes the loss-augmented inference the greatest. We need to do this since we need to find it and penalize it linearly. Then we use this $\hat{\mathbf{y}}$ to compute the gradient

$\nabla_{\mathbf{w}} L(\mathbf{w})$ by simply pluging what we found in step 1 into the formula. Later on, we are going to use this gradient to update the weights.

One thing we need to pay attention is that when we are computing the loss-augmented inference, it is actually really hard to compute since $\hat{\mathbf{y}}$ is representing all the configurations. For example if we want to output an image, then the search space becomes $\sharp\text{classes}^{\sharp\text{pixels}}$. Therefore, we need to use the methods like DP, ILP we have introduced previously to compute it.

2.4 General framework for structured learning

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

We just simply use the log-loss instead of the hinge loss in the previous SVM senario.

But right now, this framework has a limitation. The score function has to be linear. If we want our score function non-linear, we need to adjust our framework to like this.

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

$F()$ is some non-linear function, which might represent for example a deep net. So next, we need to learn how to optimize this general framework.

2.5 Optimize General Framework

We are still using gradient descent to optimize our general framework. We can see that if we want to gradient descent with respect to $\mathbf{w}$, then we can totally ignore the taskloss. Also for the simplicity, we set $\epsilon = 1$. Then we have,

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

then,

$$\nabla_{\mathbf{w}} L = C\mathbf{w} + \sum_{i\in D}(\sum_{\hat{\mathbf{y}}} \frac{expF(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}})}{\sum_{\hat{\mathbf{y}}} expF(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}})} \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) - \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \mathbf{y}^{(i)}))$$

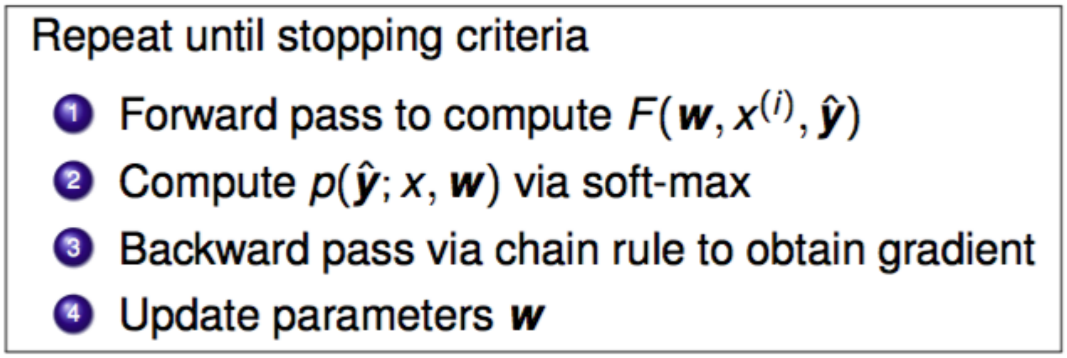
$$= C\mathbf{w} + \sum_{i\in D, \hat{\mathbf{y}}} (\frac{expF(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}})}{\sum_{\hat{\mathbf{y}}} expF(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}})} - \delta(\hat{\mathbf{y}} = \mathbf{y}^{(i)}))\nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}})$$

then we denote

$$\frac{expF(\mathbf{w},x^{(i)},\hat{\mathbf{y}})}{\sum_{\hat{\mathbf{y}}} expF(\mathbf{w},x^{(i)},\hat{\mathbf{y}})} = p(\hat{\mathbf{y}};x^{(i)},\mathbf{w})$$

we got

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



Repeat until stopping criteria

1. Forward pass to compute $F(\boldsymbol{w}, x^{(i)}, \hat{\boldsymbol{y}})$
2. Compute $p(\hat{\boldsymbol{y}}; x, \boldsymbol{w})$ via soft-max
3. Backward pass via chain rule to obtain gradient
4. Update parameters $\boldsymbol{w}$

Figure 4: Gradient descent algorithm, general

The algorithm is described in Figure 4. We noticed that $p(\hat{\mathbf{y}};x^{(i)},\mathbf{w})$ is just like the probablity weighted among the whole probablity space. We called it as the output of the soft-max layer. The question now is what we should do if the probablity space is very large or we say the output space is very large since the major factor here is $\hat{\mathbf{y}}$.

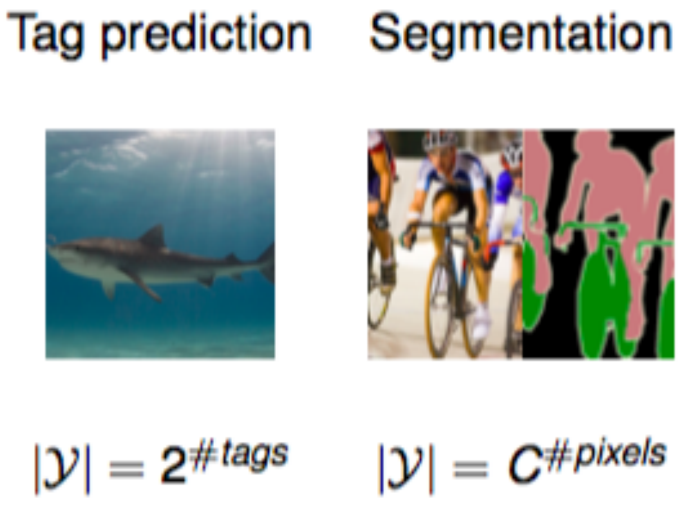


Figure 5: Output space

Again, we can use the method we introduced before, just decompose the scoring function into the sum of multiple local functions. The decomposition is totally developers' choice but we need to keep the purpose in mind that we are trying to accelerate the computation.

Therefore, we decompose it,

$$F(\mathbf{w}, x, \mathbf{y}) = \sum_r f_r(\mathbf{w}, x, \mathbf{y}_r)$$

$f_r(\mathbf{w}, x, \mathbf{y}_r)$ is some arbitrary composite function, for example a CNN. We can visualize this in Figure 6

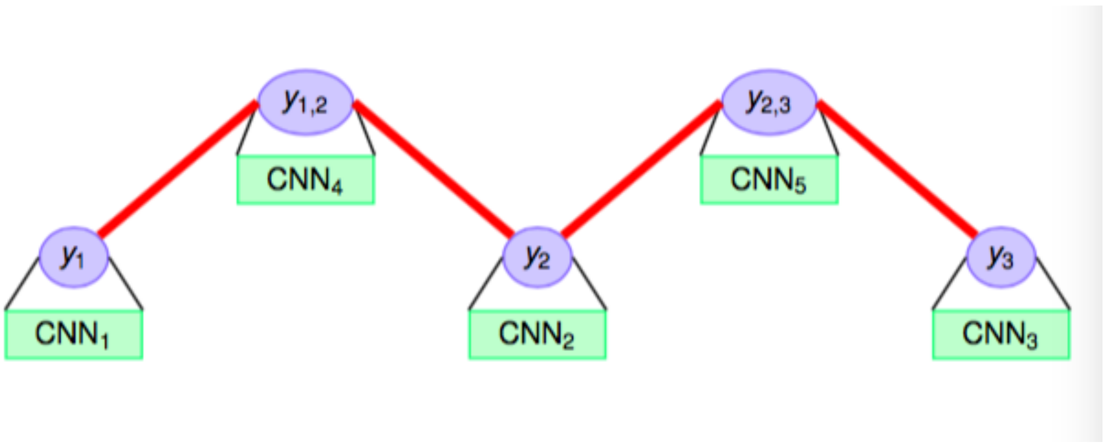


Figure 6: Structured learning with deep net

Now, we can start to compute the gradient with this decomposed scoring function.

$$\frac{\partial}{\partial \mathbf{w}} \sum_{i \in D} (log \sum_{\hat{\mathbf{y}}} exp F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) - F(\mathbf{w}, x^{(i)}, \mathbf{y}^{(i)}))$$

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

$$= \sum_{i \in D, r, \hat{\mathbf{y}}_r} (p_r(\hat{\mathbf{y}}_r; x^{(i)}, \mathbf{w}) - \delta_r(\hat{\mathbf{y}}_r = \mathbf{y}_r^{(i)})) \nabla_{\mathbf{w}} f_r(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}_r)$$

One thing here is that $p_r(\hat{\mathbf{y}}_r; x, \mathbf{w})$ cannot be computed in general way. But we can use the inference strategy to approximate it. For example we can use LP relaxation or message passing to approximate it. In the slides, we denote the approximated value as $b_r(\hat{\mathbf{y}}_r; x, \mathbf{w})$.

2.6 Examples

In the OCR task,refers to Figure 7 we want to recognize the letters in the image, we could have the first order layer to give us the local evidence and have the second order layer to have the pairwise

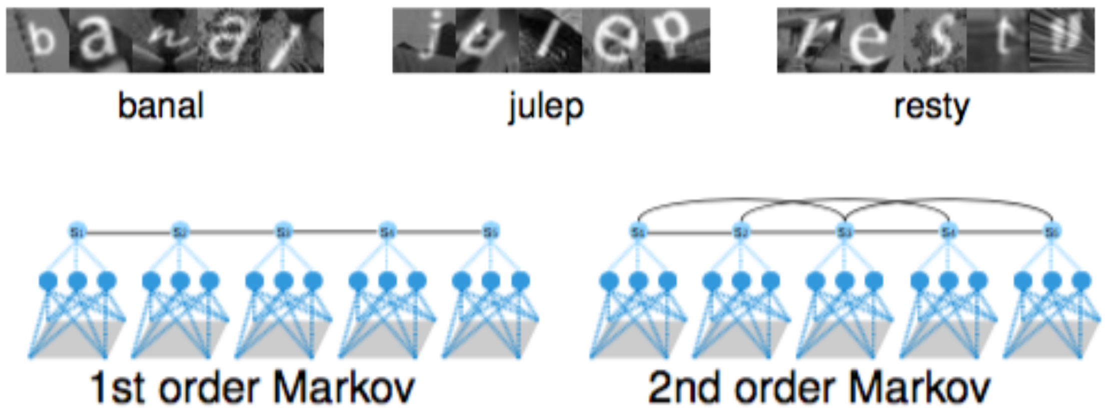


Figure 7: OCR, recognize letters in image

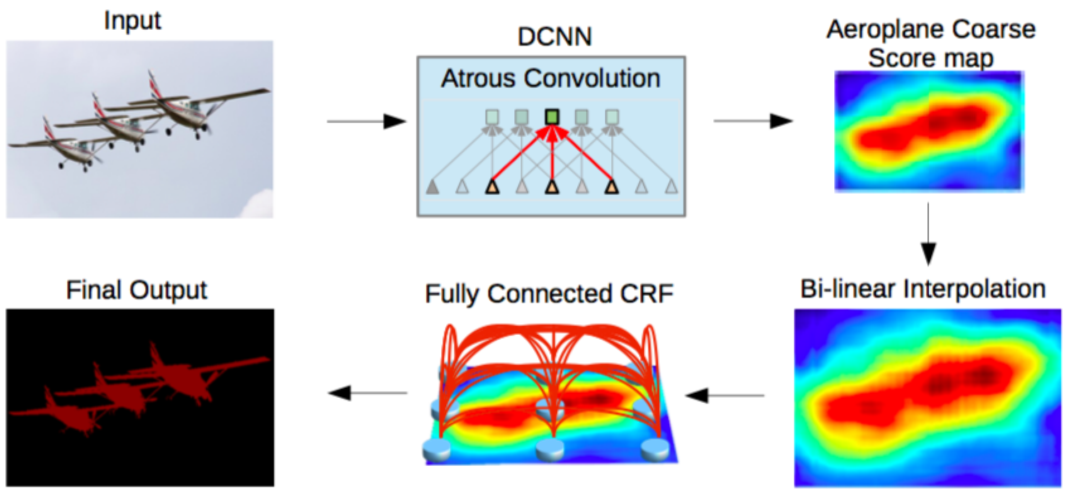


Figure 8: Semantic segmentation with deeplab [1]

evidence. Basically we just first look at each individual letters first. Then we take their relations to each other into account.

The framework proposed by deeplab on semantic segmentation task is also have the 2 order structure. The input image is firstly interact with some deep convolutional neural network, for example an AlexNet or VGG net. We get the local evidence as the coarse score map. Then it is expanded by bi-linear interpolation to the desired size of our output. Then it go through the 2nd order conditional random field to build the pairwise potentials between pixels to recover the spatial context.

3 Summary

In this lecture, we are focusing on how to fit the complex output objects into our general learning framework. We learnt how to use the techniques introduced in previous lecture and decompose the scoring function into the linear combination of multiple local functions. By this decomposition, we can greatly reduce the output space and faster our computing.

References

[1] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.