A primer on structured prediction

1 What is structured prediction?

Supervised machine learning typically aims to learn a mapping from some input space $\mathcal{X}$ to some output space $\mathcal{Y}$. In binary classification $\mathcal{Y}$ is just $\{0, 1\}$. In multi-class classification $\mathcal{X}$ is $S = \{0, \ldots, k\}$. However, for many computer vision tasks, the output space can be fairly complex. In semantic segmentation, for an $h \times w$ image, with k classes, the output space is S^{hw}, which is huge. In principle, we could consider this as just hw independent problems with an output space of S (e.g., by using convolutional network features to classify each pixel), but this independence assumption may not be justified: some combinations are inherently a lot more likely than others. For example, adjacent pixels are likely to have the same label. Such output spaces which are (a) large and (b) cannot be broken down into independent sub-problems because of underlying dependencies are often called "structured output spaces" and the problem is one of "structured prediction".

In some cases like pose estimation, the underlying dependencies may not be known before hand. In such cases, we must also learn these dependencies.

2 Energy-based models

What we want is to model the conditional probability $P(y|x)$. Under fairly general conditions, we can write $P(y|x) \propto e^{-E(y|x;\theta)}$, where E is some energy function with parameters θ. Sometimes we will write the negative of the energy as a *score function* $S(y|x; \theta)$.

2.1 Inference

Suppose we know a good energy function. How do we use it to solve the problem? We might decide to look for the most probable labeling:

$$\hat{y} = \arg\max_y P(y|x) = \arg\min_y E(y|x; \theta) = \arg\max_y S(y|x; \theta) \tag{1}$$

In a typical classification setting, the scoring function is just a score for each class, and doing this inference amounts to simply enumerating all the classes, looking at the corresponding score and picking the one with the highest score.

However, in structured prediction problems, the output space is huge and cannot be enumerated. Instead, we must consider this as an optimization problem, and look towards methods in combinatorial (for discrete output spaces) or continuous (for continuous output spaces) optimization.

Because we cannot enumerate all possible outputs and must rely on doing optimization, it follows that not all forms of the energy function will lead to tractable inference. In fact, it is often the case that one needs to severely restrict the kind of energy functions we are looking at to keep inference tractable. For example, if the output space is continuous, E needs to be convex.

Even when the optimization is tractable, it is usually an iterative process. Thus, one starts with an initialization $y^{(0)}$ and then updates it at every step. Exactly what this initialization is and how it is updated depends on the particular algorithm.

2.2 Learning

Now we want to consider how we can identify a good energy function, i.e., how we can set the parameters θ. As always, this means we want to define a *loss function.* We will consider here a particular loss function, the *margin-rescaled hinge loss* [2].

Intuitively, we want the true label y^* to score higher than every other labeling. We can try to enforce this. However, the model might then reach a solution where the true labeling scores just infinitesimally better than other labelings. Thus, we want the score of the true labeling $S(y^*|x)$ to be higher than the score of every possible labeling $S(y|x)$ *by a margin.* Structured prediction tasks have an additional property, which is that not all incorrect outputs are equally bad. For example, getting one keypoint wrong in a pose estimation problem is much better than getting all of them wrong. Let us assume that there is some function $\Delta(y, y')$ which measures the magnitude of the difference between y and y'. Then we want that wrong outputs for which $\Delta(y^*, y)$ is high should have really low scores. Thus we want that:

$$S(y|x;\theta) \leq S(y^*|x;\theta) - \Delta(y^*, y) \ \ \forall y \tag{2}$$

$$\Rightarrow S(y|x;\theta) + \Delta(y^*, y) - S(y^*|x;\theta) \leq 0 \forall y \tag{3}$$

$$\tag{4}$$

This constraint must be true for all y, which is equivalent to saying that it must be true for the y with the maximum left hand side:

$$\max_y S(y|x;\theta) + \Delta(y^*, y) - S(y^*|x;\theta) \leq 0 \tag{5}$$

We can convert this equation into a loss:

$$L(\theta, x, y^*) = \max(0, \max_y S(y|x;\theta) + \Delta(y^*, y) - S(y^*|x;\theta)) \tag{6}$$

This is the margin-rescaled hinge loss.

During training, we can sample images x and take a step along the gradient of $L(\theta, x, y^*)$. But there is a problem: the loss function involves a maximization over y, which as discussed above can be difficult because of the size of the output space. A careful look at the loss function reveals that what we actually need to maximize is the following: $S(y|x;\theta) + \Delta(y^*, y)$. This maximization looks suspiciously like the inference problem, barring the second term. If Δ is "nice" (e.g., it considers each pixel independently), then as long as we can do inference (Eq. (1)), we can do this maximization. In fact, this maximization is called "loss-augmented inference".

3 Iterated refinement

A key issue in using energy-based models is that to keep inference (and thus learning) tractable, we have to make fairly strong assumptions about the energy function. Alternatively, we have to give up hope of exact inference.

To search for an alternative, some prior work argues that in all these models, the inference is typically an iterative process, and the energy function shows up only during this iteration. For example, if the inference procedure is gradient descent, then each step in the iteration amounts to:

$$y^{(t)} \leftarrow y^{(t-1)} - \nabla_y E(y|x;\theta)\Big|_{y=y^{(t-1)}} \tag{7}$$

. Instead of first learning and energy function and then doing approximate iterated inference as above, why not directly learn the iteration. For example, we can learn a function f that does this refinement.

$$y^{(t)} \leftarrow f(y^{(t-1)}, x; \theta) \tag{8}$$

. This makes things easier because it is a straightforward feed-forward model that takes x and $y^{(t-1)}$ and can use them as it pleases.

Learning f is easy and can be done stage-wise. We only need three tuples of the form $(x, y^{(t-1)}, y^*)$, where the first two are inputs and the last is the target. Let's assume $y^{(0)}$ is some default labeling. This allows us to create an initial dataset to train f. We then use f on a new set of labeled images to get $y^{(1)}$, and create additional training tuples $(x, y^{(1)}, y^*)$. We train f on the combined dataset. We keep repeating this as much as we want.

We can also train a different f for each time step. This gives us autocontext [3] instead of inference machines [1].

The disadvantage is that we have now lost the connection to probabilistic models.

References

[1] D. Munoz. *Inference Machines: Parsing Scenes via Iterated Predictions.* PhD thesis, June 2013.

[2] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, page 104, 2004.

[3] Z. Tu. Auto-context and its application to high-level vision tasks. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.