# Machine learning for vision

Fall 2013

Roland Memisevic

Lecture 11, November 28, 2013

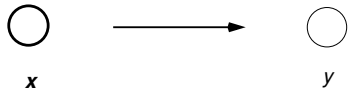


## Other ML

- Bayesian inference
- Manifold learning
- Kernel methods
- Graphical models
- Structured prediction



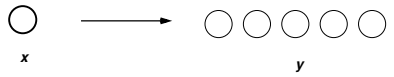
## Classification




- Classification: Given input $\boldsymbol{x}$, predict label $y$

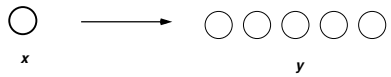


## Structured prediction




- Structured prediction: Given input $\boldsymbol{x}$, predict label *vector* $\boldsymbol{y}$
- Naive solution: Predict each component independently
- Better: Assume correlations between some labels.

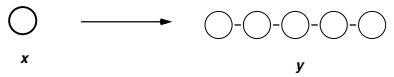
# Structured prediction




- Structured prediction: Given input $\boldsymbol{x}$, predict label *vector* $\boldsymbol{y}$
- Naive solution: Predict each component independently
- Better: Assume correlations between some labels.



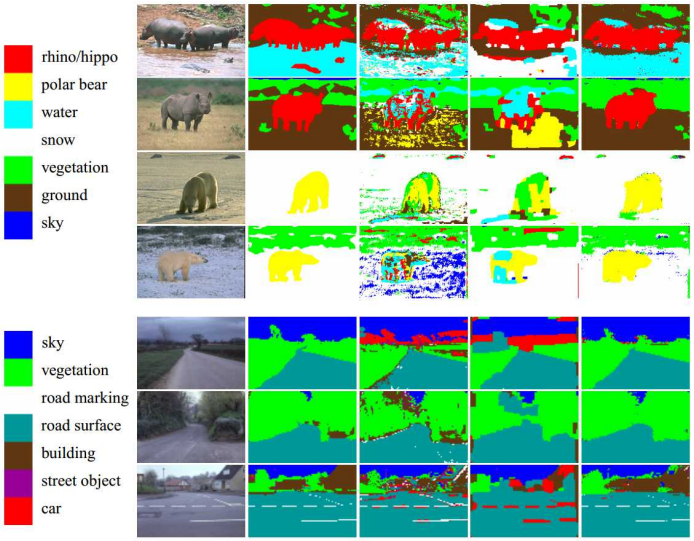
# Structured prediction




- Structured prediction: Given input $\boldsymbol{x}$, predict label *vector* $\boldsymbol{y}$
- Naive solution: Predict each component independently
- Better: Assume correlations between some labels.



# Example: scene parsing






# Linear classification with joint features

- To deal with structured outputs, it is convenient to re-formulate the classification task using a *joint feature representation* as follows:

$$f(\boldsymbol{x}) = \arg\max_{y} \ \boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{x}, y)$$

  where $\phi(\boldsymbol{x}, y)$ is a feature vector for a class/input combination.
- Learning: minimize $\frac{1}{N} \sum_i \mathrm{loss}^i(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|^2$
- We can treat standard classification in this framework, too.

## Logistic regression

- Logistic regression minimizes the "log-loss":

$$\text{loss}^i(\boldsymbol{w}) = \log \sum_y \exp\left(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i, y)\right) - \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i, y^i)$$

- This is just the log-probability over labels given inputs, defined as a softmax.



## Support vector machines

- SVMs minimize the "hinge-loss":

$$\text{loss}^i(\boldsymbol{w}) = \max_y \left(\boldsymbol{w}^T\phi(\boldsymbol{x}^i, y) + 1 - \delta_{y,y^i}\right) - \boldsymbol{w}^T\phi(\boldsymbol{x}^i, y^i)$$



## Perceptron

- Perceptrons learn by performing the updates

$$\boldsymbol{w} = \boldsymbol{w} + \eta\big(\phi(\boldsymbol{x}, y^i) - \phi(\boldsymbol{x}, \hat{y})\big)$$

where $\eta$ is a learning rate and

$$\hat{y} = \arg\max_y \boldsymbol{w}^T\phi(\boldsymbol{x}^i, y)$$



## Kernels

- Linear models can be turned into non-linear models using kernels.
- Recipe:
  1. Reformulate learning equations to use *only inner products* between data examples

$$\phi(\boldsymbol{x}_i, y_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j, y_j)$$

     and inference equations to use inner products between a test case ($\phi(\boldsymbol{x}, y)$) and training examples:

$$\phi(\boldsymbol{x}_i, y_i)^{\mathrm{T}}\phi(\boldsymbol{x}, y)$$

  2. Replace each inner product by a positive definite *kernel function*

$$k(\phi(\boldsymbol{x}_i, \boldsymbol{y}_i), \phi(\boldsymbol{x}_j, \boldsymbol{y}_j))$$

# Kernels

- One way to define positive definite kernel is to say that any matrix $K$ with entries $K_{ij} = k(\phi(\boldsymbol{x}_i, \boldsymbol{y}_i), \phi(\boldsymbol{x}_j, \boldsymbol{y}_j))$ will be positive definite (for all possible input pairs $\phi(\boldsymbol{x}_i, \boldsymbol{y}_i), \phi(\boldsymbol{x}_i, \boldsymbol{y}_i)$
- It is guaranteed (Mercer's theorem) that the kernel function corresponds to an inner product in some vector space.
- We don't care what this space is. But if it is non-linearly related to the inputs, we will have done non-linear feature extraction *implicitly*.



# Kernels

- A convex problem can be solved by alternatively solving its dual, which in this case is:

$$\max_{\alpha} \quad -\frac{1}{2\lambda}\alpha^T K\alpha + \frac{1}{\lambda}\alpha^T K\delta + \sum_i H(\alpha^i)$$
$$\text{s.t.} \quad \alpha_y^i \geq 0 \quad \forall i, y; \quad \sum_y \alpha_y^i = 1 \quad \forall i$$

  where $K_{ij} = \phi(\boldsymbol{x}_i, y_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j, y_j)$ contains all inner products between training cases.
- We may now replace the inner products by a positive definite kernel function $k(\phi(\boldsymbol{x}_i, y_i), \phi(\boldsymbol{x}_j, y_j))$, because it is guaranteed to correspond to an inner product in some space.
- **The catch:** This yields a batch learning method with complexity quadratic in the number of training cases.



# Kernels

- There are two standard ways to get the inner-product representation:
- The representer theorem: Use the fact that the learning solution can only be a weighted superposition of training examples (why?)
- Convex duality: Re-formulate the problem as a constrained convex program. For example, for logistic regression, we can write the optimization problem:

$$\min_{\boldsymbol{w},\xi} \quad \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_i \log \sum_y \exp(\xi_y^i)$$
$$\text{s.t.} \quad \xi_y^i = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i; y) - \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i; y^i) \quad \forall i, y$$



# Structured prediction

- Instead of scalar labels $y$, consider **label vectors** $\boldsymbol{y}$.
- Everything else stays the same:

$$\boldsymbol{f}(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}, \boldsymbol{y})$$

- hinge loss $->$ "structured SVM"
- log loss $->$ "conditional random field"
- perceptron rule $->$ "structured perceptron"
- But learning and inference can be intractable, because the logsumexp (for logreg) or the argmax (for SVM, perceptron) is over exponentially many $\boldsymbol{y}$

# Feature decompositions

- To regain tractability, **decompose the features** as

$$\phi(\boldsymbol{x}, \boldsymbol{y}) = \sum_s \phi(\boldsymbol{x}, \boldsymbol{y}_s)$$

where $s$ indexes a clique in some graph defined over $\boldsymbol{y}$.

- If the graph is a tree, we can tractably compute

$$\boldsymbol{f}(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \boldsymbol{w}^T \sum_s \phi(\boldsymbol{x}, \boldsymbol{y}_s)$$

as well as logsumexp's, using variations of the distributive law:



# Feature decompositions

- Consider, for example, the chain structure

$$\phi(\boldsymbol{x}, \boldsymbol{y}_s) = \phi(\boldsymbol{x}, y_t, y_{t+1})$$

$$\begin{aligned}
\boldsymbol{f}(\boldsymbol{x}) &= \arg\max_{\boldsymbol{y}} \boldsymbol{w}^T \sum_s \phi(\boldsymbol{x}, \boldsymbol{y}_s) \\
&= \arg\max_{y_1,\ldots,y_T} \boldsymbol{w}^T \sum_t \phi(\boldsymbol{x}, y_t, y_{t+1}) \\
&= \arg\max_{y_1} \ldots \arg\max_{y_T} \boldsymbol{w}^T \sum_t \phi(\boldsymbol{x}, y_t, y_{t+1}) \\
&= \arg\max_{y_1} \boldsymbol{w}^T \phi(\boldsymbol{x}, y_1, y_2) + \ldots + \arg\max_{y_T} \phi(\boldsymbol{x}, y_{T-1}, y_T)
\end{aligned}$$

- $->$ Do the argmax (or logsumexp) using dynamic programming



# Feature decompositions

- The same idea will work not just for chains but for any label vectors structured as *tree*.
- The generalization to trees is called *belief propagation*. It amounts to interpreting the intermediate computations as "messages" that are sent from node to node.
- For non-tree structures (eg. MRF), we can still sent around messages (pretending that this makes sense), and hope the values converge. Surprisingly they often do. This is called *loopy belief propagation*.
- Many other techniques for approximate inference have been developed in recent years.



# Example: Conditional Random Fields

$$p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{Z(\boldsymbol{x}; \boldsymbol{w})} \exp(\boldsymbol{w}^T \phi(\boldsymbol{x}, \boldsymbol{y}))$$

- (LeCun et al 1998, Lafferty et al. 2001).
- The clique-wise potentials $\phi(\boldsymbol{x}, \boldsymbol{y}_s)$ may be complicated non-linear functions, such as neural networks.
- Learning in that case combines belief propagation and back-propagation.
- This is possible, because these two don't interfere:

## Example: Conditional Random Fields

$$p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}) = \frac{1}{Z(\boldsymbol{x};\boldsymbol{w})}\exp(\boldsymbol{w}^T\phi(\boldsymbol{x},\boldsymbol{y}))$$

- Derivative wrt. to some parameter $\theta$:

$$\begin{aligned}\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x})}{\partial\theta} &= \boldsymbol{w}^{\mathrm{T}}\frac{\partial\phi(\boldsymbol{x},\boldsymbol{y})}{\partial\theta} - \boldsymbol{w}^{\mathrm{T}}\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})\frac{\partial\phi(\boldsymbol{x},\boldsymbol{y})}{\partial\theta} \\ &= \boldsymbol{w}^{\mathrm{T}}\sum_{s}\frac{\partial\phi(\boldsymbol{x},\boldsymbol{y}_s)}{\partial\theta} - \boldsymbol{w}^{\mathrm{T}}\sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})\frac{\partial\sum_s\phi(\boldsymbol{x},\boldsymbol{y}_s)}{\partial\theta}\end{aligned}$$

- Now think of each $\phi(\boldsymbol{x},\boldsymbol{y}_s)$ as a separate neural network that is *indexed* by $\boldsymbol{y}_s$ !
- The networks may share weights (which is trivial to implement)



## Cost-augmented inference

- "argmax"-models may have an advantage over conditional random fields, because they allows us to insert more complicated cost functions into the inference:

$$\mathrm{loss}^i(\boldsymbol{w}) = \max_{\boldsymbol{y}}\left(\boldsymbol{w}^T\phi(\boldsymbol{x}^i,\boldsymbol{y}) + \mathrm{cost}(\boldsymbol{y},\boldsymbol{y}^i)\right) - \boldsymbol{w}^T\phi(\boldsymbol{x}^i,\boldsymbol{y}^i)$$

where $\mathrm{cost}(\boldsymbol{y},\boldsymbol{y}^i)$, like the features, has to decompose in some way to be tractable.



## Kernels

- Like in the scalar label case, we can re-formulate the problem as a constrained convex program:

$$\begin{aligned}\min_{\boldsymbol{w},\xi} \quad & \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_i \log\sum_{\boldsymbol{y}}\exp(\xi^i_{\boldsymbol{y}}) \\ \text{s.t.} \quad & \xi^i_{\boldsymbol{y}} = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i;\boldsymbol{y}) - \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i;\boldsymbol{y}^i) \quad \forall i,\boldsymbol{y}\end{aligned}$$

with dual

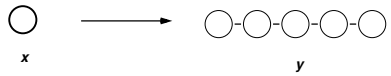
$$\begin{aligned}\max_{\alpha} \quad & -\frac{1}{2\lambda}\alpha^T K\alpha + \frac{1}{\lambda}\alpha^T K\delta + \sum_i H(\alpha^i) \\ \text{s.t.} \quad & \alpha^i_{\boldsymbol{y}} \geq 0 \quad \forall i,\boldsymbol{y}; \quad \sum_{\boldsymbol{y}}\alpha^i_{\boldsymbol{y}} = 1 \quad \forall i\end{aligned}$$



## Kernels

- For vector $\boldsymbol{y}$ this is a problem with exponentially many constraints.
- But there are ways to solve it (eg. "cutting plane method").
- Unfortunately, these methods come with quadratic complexity in the number of training case *and* quadratic complexity in the number of joint clique instantiations.
- So it seems hopeless to make them work on large datasets.

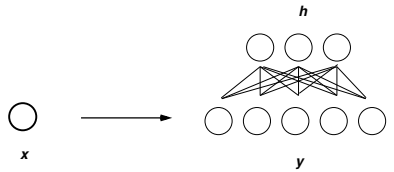
# Output representation learning




- A possible alternative to complicated graphical models on the outputs is to use *feature learning* on the outputs themselves.



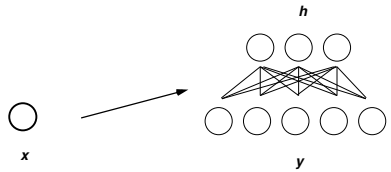
# Output representation learning




- A possible alternative to complicated graphical models on the outputs is to use *feature learning* on the outputs themselves.



# Output representation learning




- A possible alternative to complicated graphical models on the outputs is to use *feature learning* on the outputs themselves.
- (...and potentially the dependence of the features themselves on the inputs, which would require 3-way connections)



# Feature learning MRFs

- MRFs defined via feature learning have been getting quite popular in vision, because they perform very well in denoising and inpainting tasks.
- For example, *Fields of Experts* (Roth & Black), which are convolutional RBMs.
- The free energy itself is a sum over the individual clique free energies, and it can be written as a convolution.
- Same for the derivative of the free energy wrt. the input image. (What is it ?!?)
- So it is easy to optimize it to perform inference.

# Feature learning MRFs

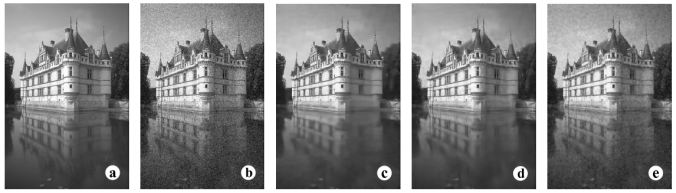


Figure 4. **Denoising results.** **(a)** Original noiseless image. **(b)** Image with additive Gaussian noise ($\sigma = 25$); PSNR $= 20.29$dB. **(c)** Denoised image using a Field of Experts; PSNR $= 28.72$dB. **(d)** Denoised image using the approach from [20]; PSNR $= 28.90$dB. **(e)** Denoised image using standard non-linear diffusion; PSNR $= 27.18$dB.

Figure 6. **Inpainting with a Field of Experts.** **(a)** Original image with overlaid text. **(b)** Inpainting result from diffusion using the FoE prior. **(c)** Close-up comparison between **a** (left), **b** (middle), and the results from [3] (right).



# Other stuff

- Saliency, attention
- Other modalities
- Geometry (not much learning, but some starting now with complex cells)
- 3d models, shape
- models of humans, MOCAP