CPSC 540: Machine Learning

Conditional Random Fields

Mark Schmidt

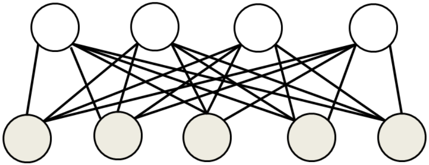
University of British Columbia

Winter 2018

Last Time: Restricted Boltzmann Machines

- We discussed restricted Boltzmann machines as mix of clustering/latent-factors,

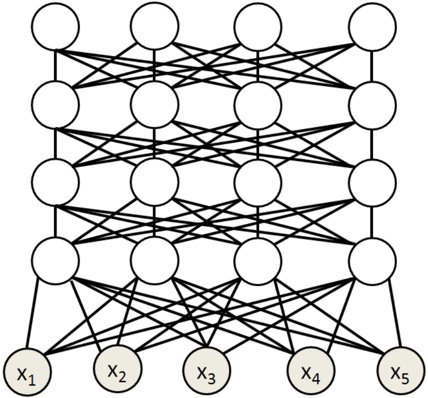
$$p(x,z) \propto \left(\prod_{j=1}^{d} \phi_j(x_j)\right)\left(\prod_{c=1}^{k} \phi_c(z_c)\right)\left(\prod_{j=1}^{d}\prod_{c=1}^{k} \phi_{jc}(x_j,z_c)\right).$$



- Bipartite structure allows block Gibbs sampling:
  - Conditional UGM removes observed nodes.
  - Training by alternating between stochastic gradient and Gibbs updates.
- Ingredient for training deep belief networks: started deep learning movement.

Deep Boltzmann Machines

- Deep Boltzmann machines just keep as an undirected model.
  - Sampling is nicer: no explaning away within layers.
  - Variables in layer are independent given variables in layer above and below.



Deep Boltzmann Machines

- Performance of deep Boltzmann machine on NORB data:

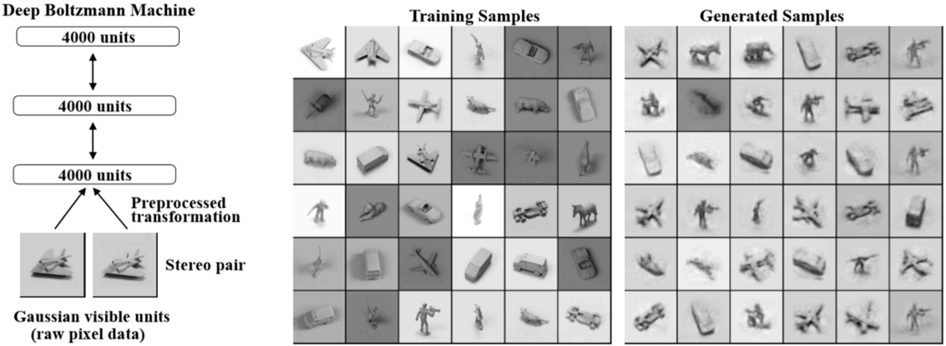
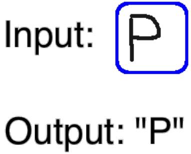


Figure 5: **Left:** The architecture of deep Boltzmann machine used for NORB. **Right:** Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

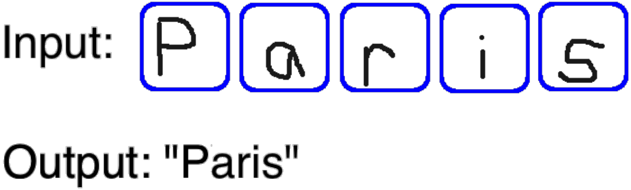
http://www.cs.toronto.edu/~fritz/absps/dbm.pdf

Motivation: Structured Prediction

Classical supervised learning focuses on predicting single discrete/continuous label:



Structured prediction allows general objects as labels:



Above output is a word, but it could be sequence/molecule/image/PDF.

3 Classes of Structured Prediction Methods

3 main approaches to structured prediction:

1. Generative models use $p(y \mid x) \propto p(y, x)$ as in naive Bayes.
   - Turns structured prediction into density estimation.
     - But remember how hard it was just to model images of digits?
     - We have to model features and solve supervised learning problem.
2. Discriminative models directly fit $p(y \mid x)$ as in logistic regression.
   - View structured prediction as conditional density estimation.
     - Just focuses on modeling y given x, not trying to modle features x.
     - Lets you use complicated features x that make the task easier.
3. Discriminant functions just try to map from x to y as in SVMs.
   - Now you don't even need to worry about calibrated probabilities.

Outline

1. Conditional Random Fields
2. Neural Networks Review
3. Structured Support Vector Machines

Rain Data without Month Information

- Consider an Ising model for the rain data with tied parameters,

$$p(y_1, y_2, \dots, y_k) \propto \exp\left(\sum_{c=1}^{k} y_c w + \sum_{c=2}^{k} y_c y_{c-1} v\right).$$

- First term reflects that "not rain" is more likely.
- Second term reflects that consecutive days are more likely to be the same.

- But how can we model that "some months are less rainy"?

Rain Data with Month Information: Boltzmann Machine

- We could add 12 binary latent variable z_j,

$$p(y_1, y_2, \dots, y_k, z) \propto \exp\left(\sum_{c=1}^{k} y_c w + \sum_{c=2}^{k} y_c y_{c-1} v + \sum_{c=1}^{k}\sum_{j=1}^{12} y_c z_j v_j + \sum_{j=1}^{12} z_j w_j\right),$$

  which is a Boltzmann machine.
  - Modifies the probability of "rain" for each of the 12 values.

- Inference is more expensive due to the extra variables.
  - Learning is also non-convex since we need to sum over z.

Rain Data with Month Information: MRF

- If we *know the months* we just could add an explicit month feature x_j

$$p(y_1, y_2, \dots, y_k, x) \propto \exp\left(\sum_{c=1}^{k} y_c w + \sum_{c=2}^{k} y_c y_{c-1} v + \sum_{c=1}^{k}\sum_{j=1}^{12} y_c x_j v_j + \sum_{j=1}^{12} x_j w_j\right),$$

- Learning might be easier: we're given known clusters.
- But still have to model distribution x, and density estimation isn't easy.
  - It's easy in this case because months are uniform.
  - But in other cases we may want to use a complicated x.
  - And inference is more expensive than chain-structured models.

Rain Data with Month Information: CRF

- In conditional random fields we fit distribution conditioned on features x,

$$p(y_1, y_2, \dots, y_k \mid x) = \frac{1}{Z(x)} \exp\left(\sum_{c=1}^{k} y_c w + \sum_{c=2}^{d} y_c y_{c-1} v + \sum_{c=1}^{k}\sum_{j=1}^{12} y_c x_j v_j\right).$$

  - Now we don't need to model x.
    - Just need to figure out how x affects y.
  - This is like logistic regression (no model of x) instead of naive Bayes (modeling x).
    - $p(y \mid x)$ (discriminative) vs. $p(y, c)$ (generative).

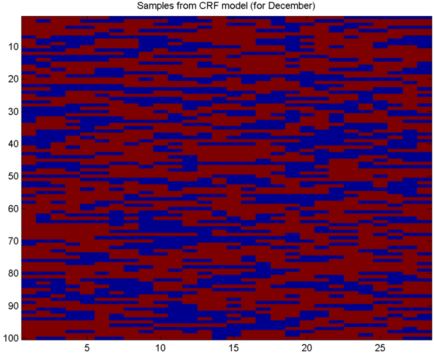
- The conditional UGM given x has a chain-structure

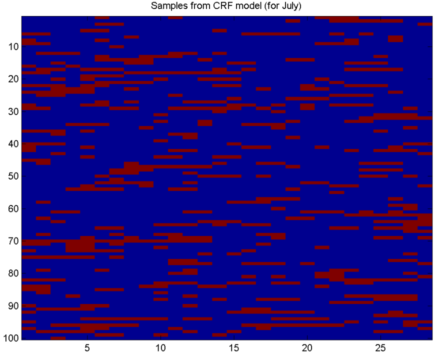
$$\phi_i(y_i) = \exp\left(y_i w + \sum_{j=1}^{12} y_i x_j v_j\right), \quad \phi_{ij}(y_i, y_j) = \exp(y_i y_j v),$$

  so inference can be done using forward-backward.
  - And it's log-linear so the NLL will be convex.

Rain Data with Month Information

- Samples from CRF conditioned on x for December and July:

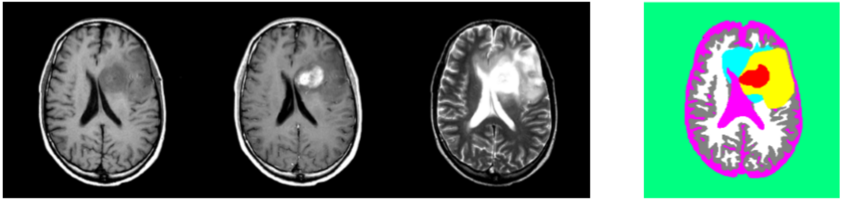




- Code available as part of UGM package.

Motivation: Automatic Brain Tumor Segmentation

- Task: identification of tumours in multi-modal MRI.



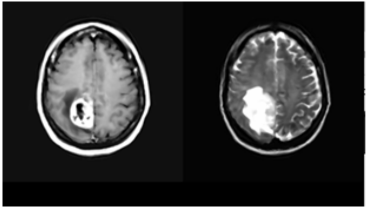
- Applications:
  - Radiation therapy target planning, quantifying treatment response.
  - Mining growth patterns, image-guided surgery.
- Challenges:
  - Variety of tumor appearances, similarity to normal tissue.
  - “You are never going to solve this problem”.

Brain Tumour Segmentation with Label Dependencies

- After a lot pre-processing and feature engineering (convolutions, priors, etc.), final system used logistic regression to label each pixel as "tumour" or not.

$$p(y_c \mid x_c) = \frac{1}{1 + \exp(-y_c w^T x_c)} = \frac{\exp(y_c w^T x_c)}{\exp(w^T x_c) + \exp(-w^T x_c)}$$

- Gives a high "pixel-level" accuracy, but sometimes gives silly results:



- Classifying each pixel independently misses dependence in labels y^i:
  - We prefer neighbouring voxels to have the same value.

Brain Tumour Segmentation with Label Dependencies

- With independent logistic, joint distribution over all labels in one image is

$$p(y_1, y_2, \dots, y_k \mid x_1, x_2, \dots, x_k) = \prod_{c=1}^{k} \frac{\exp(y_c w^T x_c)}{\exp(w^T x_c) + \exp(-w^T x_c)}$$
$$\propto \exp\left(\sum_{c=1}^{d} y_c w^T x_c\right),$$

where here x_c is the feature vector for position c in the image.

- We can view this is a log-linear UGM with no edges,

$$\phi_c(y_c) = \exp(y_c w^T x_c),$$

so given the x_c there is no dependence between the y_c.

Brain Tumour Segmentation with Label Dependencies

- Adding an Ising-like term to model dependencies between y_i gives

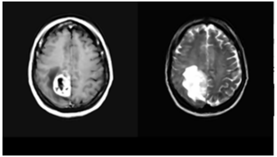
$$p(y_1, y_2, \dots, y_k \mid x_1, x_2, \dots, x_k) \propto \exp\left(\sum_{c=1}^{k} y_c w^T x_c + \sum_{(c,c') \in E} y_c y_{c'} v\right),$$

- Now we have the same "good" logistic regression model, but v controls how strongly we want neighbours to be the same.

- Note that we're going to jointly learn w and v.
  - We'll find the optimal joint logistic regression and Ising model.

Conditional Random Fields for Segmentation

- Recall the performance with the independent classifier:



- The pairwise CRF better modelled the "guilt by association":

(We were using edge features $x_{cc'}$ too, see bonus.)

Conditional Random Fields

- The [b]rain CRF can be written as a conditional log-linear models,

$$p(y \mid x, w) = \frac{1}{Z(x)} \exp(w^T F(x, y)),$$

for some parameters w and features $F(x, y)$.

- The NLL is convex and has the form

$$-\log p(y \mid x, w) = -w^T F(x, y) + \log Z(x),$$

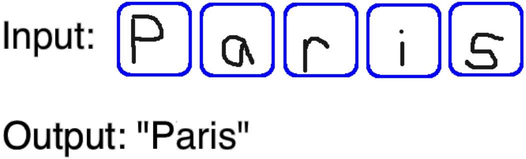
and the gradient can be written as

$$-\nabla \log p(y \mid x, w) = -F(x, y) + \mathbb{E}_{y \mid x}[F(x, y)].$$

- Unlike before, we now have a $Z(x)$ and set of marginals for each x.
  - Train using gradient methods like quasi-Newton, SG, or SAG.

Modeling OCR Dependencies

- What dependencies should we model for this problem?



- $\phi(y_c, x_c)$: potential of individual letter given image.
- $\phi(y_{c-1}, y_c)$: dependency between adjacent letters ('q-u').
- $\phi(y_{c-1}, y_c, x_{c-1}, x_c)$: adjacent letters and image dependency.
- $\phi_c(y_{i-1}, y_c)$: inhomogeneous dependency (French: 'e-r' ending).
- $\phi_c(y_{c-2}, y_{c-1}, y^i)$: third-order and inhomogeneous (English: 'i-n-g' end).
- $\phi(y \in \mathcal{D})$: is y in dictionary $\mathcal{D}$?

Tractability of Discriminative Models

- Features can be very complicated, since we just condition on the x_c, .
- Given the x_c, tractability depends on the conditional UGM on the y_c.
  - Inference/decoding will be fast or slow, depending on the y_c graph.
- Besides "low treewidth", some other cases where exact computation is possible:
  - Semi-Markov chains (allow dependence on time you spend in a state).
  - Context-free grammars (allows potentials on recursively-nested parts of sequence).
  - Sum-product networks (restrict potentials to allow exact computation).
  - "Dictionary" feature is non-Markov, but exact computation still easy.
- We can alternately use our previous approximations:
  1. Pseudo-likelihood (what we used).
  2. Monte Carlo approximate inference (better but slower).
  3. Variational approximate inference (fast, quality varies).

Learning for Structured Prediction

3 types of classifiers discussed in CPSC 340/540:

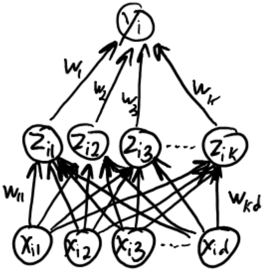
Model	"Classic ML"	Structured Prediction
Generative model $p(y, x)$	Naive Bayes, GDA	UGM (or MRF)
Discriminative model $p(y \mid x)$	Logistic regression	CRF
Discriminant function $y = f(x)$	SVM	Structured SVM

- Discriminative models don't need to model x.
  - Don't need "naive Bayes" or Gaussian assumptions.
- Discriminant functions don't even worry about probabilities.
  - Based on decoding, which is different than inference in structured case.
- See bonus slides for previous lecture material on structured SVMs.
  - Useful when inference is hard but decoding is easy ("attractive models").

Outline

Feedforward Neural Networks

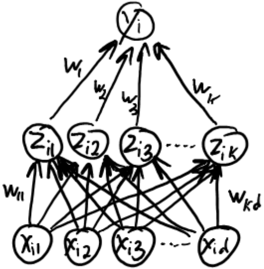
- In 340 we discussed feedforward neural networks for supervised learning.
- With 1 hidden layer the classic model has this structure:



- Motivation:
  - For some problems it's hard to find good features.
  - This learn features z that are good for supervised learning.

Neural Networks as DAG Models

- It's a DAG model but there is an important difference with our previous models:
  - The latent variables z_c are deterministic functions of the x_j.



- Makes inference given x trivial: if you observe all x_j you also observe all z_c.
  - In this case y is the only random variable.

Neural Network Notation

- We'll continue using our supervised learning notation:

$$X = \begin{bmatrix} \text{———} & (x^1)^T & \text{———} \\ \text{———} & (x^2)^T & \text{———} \\ & \vdots & \\ \text{———} & (x^n)^T & \text{———} \end{bmatrix}, \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix},$$

- For the latent features and two sets of parameters we'll use

$$Z = \begin{bmatrix} \text{———} & (z^1)^T & \text{———} \\ \text{———} & (z^2)^T & \text{———} \\ & \vdots & \\ \text{———} & (z^n)^T & \text{———} \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}, \quad W = \begin{bmatrix} \text{———} & w_1 & \text{———} \\ \text{———} & w_2 & \text{———} \\ & \vdots & \\ \text{———} & w_k & \text{———} \end{bmatrix},$$

where Z is n by k and W is k by d.

Introducing Non-Linearity

- We discussed how the "linear-linear" model,

$$z^i = Wx^i, \quad y^i = v^T z^i,$$

is degenerate since it's still a linear model.

- The classic solution is to introduce a non-linearity,

$$z^i = h(Wx^i), \quad y^i = v^T z^i,$$

where a common-choice is applying sigmoid element-wise,

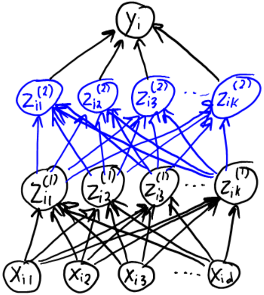
$$z_c^i = \frac{1}{1 + \exp(-w_c x^i)},$$

which is said to be the "activation" of neuron c on example i.

  - A universal approximator with k large enough.

Deep Neural Networks

- In deep neural networks we add multiple hidden layers,

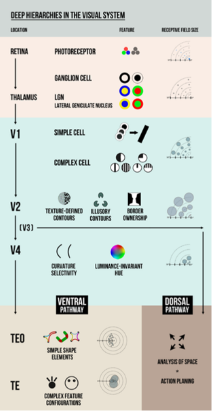


- Mathematically, with 3 hidden layers the classic model uses

$$y^i = v^T \underbrace{h(W^3 \underbrace{h(W^2 \underbrace{h(W^1 x^i)}_{z^{i1}})}_{z^{i2}})}_{z^{i3}}.$$

Biological Motivation

- Deep learning is motivated by theories of deep hierarchies in the brain.



https://en.wikibooks.org/wiki/Sensory_Systems/Visual_Signal_Processing

- But most research is about making models work better, not be more brain-like.

Deep Neural Network History

- Popularity of deep learning has come in waves over the years.
  - Currently, it is one of the hottest topics in science.
- Recent popularity is due to unprecedented performance on some difficult tasks:
  - Speech recognition.
  - Computer vision.
  - Machine translation.
- These are mainly due to big datasets, deep models, and tons of computation.
  - Plus some tweaks to the classic models.
- For a NY Times article discussing some of the history/successes/issues, see:

`https://mobile.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html`

Summary

- 3 types of structured prediction:
  - Generative models, discriminative models, discriminant functions.
- Conditional random fields generalize logistic regression:
  - Discriminative model allowing dependencies between labels.
  - Log-linear parameterization again leads to convexity.
  - But requires inference in graphical model.
- Neural networks learn features for supervised learning.
- Next time: modern convolutional neural networks and applications.
  - Image segmentation, depth estimation, image colorization.

Brain Tumour Segmentation with Label Dependencies

- We got a bit more fancy and used edge features x^{ij},

$$p(y^1, y^2, \dots, y^d \mid x^1, x^2, \dots, x^d) = \frac{1}{Z} \exp\left(\sum_{i=1}^{d} y^i w^T x^i + \sum_{(i,j) \in E} y^i y^j v^T x^{ij}\right).$$

- For example, we could use $x^{ij} = 1/(1 + |x^i - x^j|)$.
  - Encourages y_i and y_j to be more similar if x^i and x^j are more similar.

- This is a pairwise UGM with

$$\phi_i(y^i) = \exp(y^i w^T x^i), \quad \phi_{ij}(y^i, y^j) = \exp(y^i y^j v^T x^{ij}),$$

so it didn't make inference any more complicated.

Outline

SVMs and Likelihood Ratios

- Logistic regression optimizes a likelihood of the form

$$p(y^i \mid x^i, w) \propto \exp(y^i w^T x^i).$$

- But if we only want correct decisions it's sufficient to have

$$\frac{p(y^i \mid x^i, w)}{p(-y^i \mid x^i, w)} \geq \kappa,$$

  for any $\kappa > 1$.
- Taking logarithms and plugging in probabilities gives

$$y^i w^T x^i + \log Z - (-y^i w^T x^i) - \log Z \geq \log \kappa$$

- Since κ is arbitrary let's use $\log(\kappa) = 2$,

$$y^i w^T x^i \geq 1.$$

SVMs and Likelihood Ratios

- So to classify all i correctly it's sufficient that

$$y^i w^T x^i \geq 1,$$

but this linear program may have no solutions.

- To give solution, allow non-negative "slack" r_i and penalize size of r_i,

$$\underset{w,r}{\text{argmin}} \sum_{i=1}^{n} r_i \quad \text{with} \quad y^i w^T x^i \geq 1 - r_i \quad \text{and} \quad r_i \geq 0.$$

- If we apply our Day 2 linear programming trick in reverse this minimizes

$$f(w) = \sum_{i=1}^{n} [1 - y^i w^T x^i]^+$$

and adding an L2-regularizer gives the standard SVM objective.
  - The notation $[\alpha]^+$ means $\max\{0, \alpha\}$.

Multi-Class SVMs: nk-Slack Formulation

- With multi-class logistic regression we use

$$p(y^i = c \mid x^i, w) \propto \exp(w_c^T x^i).$$

- If want correct decisions it's sufficient for all $y' \neq y^i$ that

$$\frac{p(y^i \mid x^i, w)}{p(y' \mid x^i, w)} \geq \kappa.$$

- Following the same steps as before, this corresponds to

$$w_{y^i}^T x^i - w_{y'}^T x^i \geq 1.$$

- Adding slack variables our linear programming trick gives

$$f(W) = \sum_{i=1}^{n} \sum_{y' \neq y^i} [1 - w_{y^i}^T x^i + w_{y'}^T x^i]^+,$$

which with L2-regularization we'll call the nk-slack multi-class SVM.

Multi-Class SVMs: n-Slack Formulation

- If we want correct decisions it's also sufficent that

$$\frac{p(y^i \mid x^i, w)}{\max_{y' \neq y^i} p(y' \mid x^i, w)}.$$

- This leads to the constraints

$$\max_{y' \neq y^i} \{w_{y^i}^T x^i - w_{y'}^T x^i\} \geq 1.$$

- Following the same steps gives an alternate objective

$$f(W) = \sum_{i=1}^{n} \max_{y' \neq y^i} [1 - w_{y^i}^T x^i + w_{y'}^T x^i]^+,$$

which with L2-regularization we'll call the n-slack multi-class SVM.

Multi-Class SVMs: nk-Slack vs. n-Slack

- Our two formulations of multi-class SVMs:

$$f(W) = \sum_{i=1}^{n} \sum_{y' \neq y^i} [1 - w_{y^i}^T x^i + w_{y'}^T x^i]^+ + \frac{\lambda}{2}\|W\|_F^2,$$

$$f(W) = \sum_{i=1}^{n} \max_{y' \neq y^i} [1 - w_{y^i}^T x^i + w_{y'}^T x^i]^+ + \frac{\lambda}{2}\|W\|_F^2.$$

- The nk-slack loss penalizes based on all y' that could be confused with y^i.
- The n-slack loss only penalizes based on the "most confusing" alternate example.

- While nk-slack often works better, n-slack can be used for structured prediction...

Hidden Markov Support Vector Machines

- For decoding in conditional random fields to get the entire labeling correct we need

$$\frac{p(y^i \mid x^i, w)}{p(y' \mid x^i, w)} \geq \gamma,$$

for all alternative configuraitons y'.

- Following the same steps are before we obtain

$$f(w) = \sum_{i=1}^{n} \max_{y' \neq y} [1 - \log p(y^i \mid x^i, w) + \log p(y' \mid x^i, w)]^+ + \frac{\lambda}{2}\|w\|^2,$$

the hidden Markov support vector machine (HMSVM).

- Tries to make log-probability of true y^i greater than for other y' by more than 1.

Hidden Markov Support Vector Machines

- Two problems with the HMSVM:
  1. It requires finding second-best decoding, which is harder than decoding.
  2. It views any alternative labeling y' as equally bad.

- Suppose that $y^i = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}$, and predictions of two models are

$$y' = \begin{bmatrix}1 & 1 & 0 & 1\end{bmatrix}, \quad y' = \begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix},$$

  should both models receive the same loss on this example?

Adding a Loss Function

- We can fix both HMSVM issues by replacing the "correct decision" constraint,

$$\log p(y^i \mid x^i, w) - \log p(y' \mid x^i, w) \geq 1,$$

with a constraint containing a loss function g,

$$\log p(y^i \mid x^i, w) - \log p(y' \mid x^i, w) \geq g(y^i, y').$$

- Usually we take $g(y^i, y')$ to be the difference between y^i and y'.
- If $g(y^i, y^i) = 0$, you can maximize over all y' instead of $y' \neq y^i$.
  - Further, if g is written as sum of functions depending on the graph edges, finding "most violated" constraint is equivalent to decoding.

Structured SVMs

- These constraints lead to the max-margin Markov network objective,

$$f(w) = \sum_{i=1}^{n} \max_{y'}[g(y^i, y') - \log p(y^i \mid x^i, w) + \log p(y' \mid x^i, w)]^+ + \frac{\lambda}{2}\|w\|^2,$$

  which is also known as a structured SVM.

- Beyond learning principle, key differences between CRFs and SSVMs:
  - SSVMs require decoding, not inference, for learning:
    - Exact SSVMs in cases like graph cuts, matchings, rankings, etc.
  - SSVMs have loss function for complicated accuracy measures:
    - But loss needs to decompose over parts for tractability.
    - Could also formulate 'loss-augmented' CRFs.

- We can also train with approximate decoding methods.
  - State of the art training: block-coordinate Frank Wolfe (bonus slides).

SVMs for Ranking with Pairwise Preference

- Suppose we want to rank examples.
- A common setting is with features x^i and pairwise preferences:
  - List of objects (i, j) where we want $y^i > y^j$.
- Assuming a log-linear model,

$$p(y^i \mid x^i, w) \propto \exp(w^T x^i),$$

we can derive a loss function based on the pairwise preference decisiosn,

$$\frac{p(y^i \mid x^i, w)}{p(y^j \mid x^j, w)} \geq \gamma,$$

which gives a loss function of the form

$$f(w) = \sum_{(i,j) \in R} [1 - w^T x^i + w^T x^j]^+.$$

Fitting Structured SVMs

Overview of progress on training SSVMs:

- Cutting plane and bundle methods (e.g., svmStruct software):
  - Require $O(1/\epsilon)$ iterations.
  - Each iteration requires decoding on every training example.
- Stochastic sub-gradient methods:
  - Each iteration requires decoding on a single training example.
  - Still requires $O(1/\epsilon)$ iterations.
  - Need to choose step size.
- Dual Online exponentiated gradient (OEG):
  - Allows line-search for step size and has $O(1/\epsilon)$ rate.
  - Each iteration requires inference on a single training example.
- Dual block-coordinate Frank-Wolfe (BCFW):
  - Each iteration requires decoding on a single training example.
  - Requires $O(1/\epsilon)$ iterations.
  - Closed-form optimal step size.
  - Theory allows approximate decoding.

Block Coordinate Frank Wolfe

Key ideas behind BCFW for SSVMs:

- Dual problem has as the form

$$\min_{\alpha_i \in \mathcal{M}_i} F(\alpha) = f(A\alpha) - \sum_i f_i(\alpha_i).$$

  where f is smooth.
- Problem structure where we can use block coordinate descent:
  - Normal coordinate updates intractable because $\alpha_i \in |\mathcal{Y}|$.
  - But Frank-Wolfe block-coordinate update is equivalent to decoding

$$s = \operatorname*{argmin}_{s' \in \mathcal{M}_i} F(\alpha) + \langle \nabla_i F(\alpha), s' - \alpha_i \rangle.$$

$$\alpha_i = \alpha_i - \gamma(s - \alpha_i).$$

  - Can implement algorithm in terms of primal variables.
- Connections between Frank-Wolfe and other algorithms:
  - Frank-Wolfe on dual problem is subgradient step on primal.
  - 'Fully corrective' Frank-Wolfe is equivalent to cutting plane.