

CPSC 440/540: Advanced Machine Learning

Log-Linear Models, CRFs, and Mixtures

Danica Sutherland (building on materials from Mark Schmidt)

University of British Columbia

Winter 2023

Last Time: Undirected Graphical Models

- We discussed **undirected graphical models**

$$p(x_1, x_2, \dots, x_d) \propto \prod_{c \in \mathcal{C}} \phi_c(x_c),$$

which write joint distribution as product of non-negative potentials over subsets c .

- The most common variant is **pairwise UGMs**,

$$p(x_1, x_2, \dots, x_d) \propto \left(\prod_{j=1}^d \phi_j(x_j) \right) \left(\prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j) \right),$$

which includes Markov chains and multivariate Gaussians as special cases.

- In tree-structured graphs (**no loops**), common inference operations are $O(dk^2)$.
 - By generalizing the methods used for Markov chains.
 - But runtime is **exponential in “treewidth”** for graphs with loops.

Vancouver Rain Data: DAG vs. UGM

- We previously considered the “Vancouver Rain” dataset:

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	...
Month 1	0	0	0	1	1	0	0	1	1	
Month 2	1	0	0	0	0	0	1	0	0	
Month 3	1	1	1	1	1	1	1	1	1	
Month 4	1	1	1	1	0	0	1	1	1	
Month 5	0	0	0	0	1	1	0	0	0	
Month 6	0	1	1	0	0	0	0	1	1	

- We previously fit this with a Markov chain under the DAG factorization:

$$p(x_1, x_2, \dots, x_d) = p(x_1) \prod_{j=2}^d p(x_j \mid x_{j-1}),$$

where we used tabular potentials (so learning was counting).

Vancouver Rain Data: DAG vs. UGM

- Consider fitting a Markov chain under a UGM factorization:

$$p(x_1, x_2, \dots, x_d) \propto \left(\prod_{j=1}^d \phi_j(x_j) \right) \left(\prod_{j=2}^d \phi_{j,j-1}(x_j, x_{j-1}) \right).$$

- Consider the following UGM parameterization (for $x_j \in \{-1, +1\}$):

$$\phi_j(x_j) = \exp(w_j x_j), \quad \phi_{ij}(x_i, x_j) = \exp(v_{ij} x_i x_j),$$

where w_j is a node weight, v_{ij} is an edge weight, and we have used **Ising** edges.

- The exponential function makes the potentials non-negative.
 - We call this a **log-linear** model: logarithms of potentials are linear.
- Ising potentials can reflect **how strongly neighbours are attracted/repulsed**.
- For the rain data, we would expect $v_{ij} > 0$ (adjacent days likely to have same value).
- For the rain data, it makes sense to **tie w_j across j and v_{ij} across (i, j) values**.

Vancouver Rain Data: DAG vs. UGM

- Our log-linear model of the rain data under the Ising parameterization:

$$\begin{aligned} p(x_1, x_2, \dots, x_d \mid w, v) &\propto \left(\prod_{j=1}^d \exp(wx_j) \right) \left(\prod_{j=2}^d \exp(vx_j x_{j-1}) \right) \\ &= \exp \left(\sum_{j=1}^d wx_j + \sum_{j=2}^d vx_j x_{j-1} \right) \\ &= \exp \left(w \sum_{j=1}^d x_j + v \sum_{j=2}^d x_j x_{j-1} \right) \\ &= \exp \left(\begin{bmatrix} w \\ v \end{bmatrix}^\top \begin{bmatrix} \sum_{j=1}^d x_j \\ \sum_{j=2}^d x_j x_{j-1} \end{bmatrix} \right). \end{aligned}$$

- This is an **exponential family** in canonical form!
 - NLL will be convex in terms of w and v ; derivative of NLL has simple form.
 - If we didn't tie parameters, we'd have a statistic for each time.

Learning Log-Linear Model for Vancouver Rain Data

- Canonical form: $p(x | w, v) \propto \exp \left(\begin{bmatrix} w \\ v \end{bmatrix}^\top \begin{bmatrix} \sum_{j=1}^d x_j \\ \sum_{j=2}^d x_j x_{j-1} \end{bmatrix} \right)$.
 - Sufficient statistics $s_1(x) = \sum_{j=1}^d x_j$, $s_2(x) = \sum_{j=2}^d x_j x_{j-1}$.
- We derived in general for **canonical-form exponential families** that

$$\nabla_{\theta}[-\log p(\mathbf{X} | \theta)] = -\sum_{i=1}^n s(x^i) + n \mathbb{E}[s(X) | \theta].$$

- Can't solve analytically here. . . but we can just **run gradient descent!**
- We have $\mathbb{E}[s(X) | w, v] = \begin{bmatrix} \sum_{j=1}^d 2 (\Pr(X_j = 1 | w, v) - 1) \\ \sum_{j=2}^d (2 \Pr(X_j = X_{j-1} | w, v)) \end{bmatrix}$.
 - Can compute **all of these marginals** with **forward-backward**.
 - Could also compute $\log Z$ and use autodiff.

Learning Log-Linear Models (In General)

- We often write **log-linear** UGMs in an exponential family form

$$p(x | w) = \frac{\exp(w^\top F(x))}{Z(w)},$$

where the **feature functions** $F(x)$ count the number of times we use each w_j .

- Examples of feature functions, and potentials for categoricals, in **bonus slides**.
- Feature functions are just sufficient statistics, so

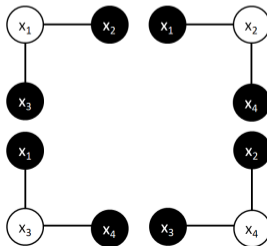
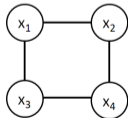
$$\nabla_w[-\log p(\mathbf{X} | w)] = -\sum_{i=1}^n F(x^i) + n \mathbb{E}[s(X) | w].$$

- Computing this **requires inference**, which is #P-hard in general graphs.
 - So we need to consider **approximations when learning**.

Approximate Learning: Pseudo-Likelihood

- A popular approximation to the NLL is **pseudo-likelihood**.
 - “Fast, convex, and crude.”
- Pseudo-likelihood turns learning into d single-variables problem (similar to DAGs),

$$p(x_1, x_2, \dots, x_d) \approx \prod_{j=1}^d p(x_j | x_{-j}) = \prod_{j=1}^d p(x_j | x_{\text{nei}(j)}).$$



Approximate Learning: Marginal Approximations

- Another way to approximate the NLL is with **approximate inference**.
 - ① Deterministic **variational approximations** of $\mathbb{E}[F(x)]$ (more on these later).
 - Approximate p by a simpler q , and compute expectation for q .
 - ② **Monte Carlo** approximation of $\mathbb{E}[F_j(x)]$ given current parameters w :

$$\begin{aligned}\nabla f(w) &= -F(\mathbf{X}) + \mathbb{E}[F(x)] \\ &\approx -F(\mathbf{X}) + \underbrace{\frac{1}{t} \sum_{i=1}^t F(x^i)}_{\text{Monte Carlo approx}},\end{aligned}$$

based on samples from $p(x | w)$.

- Unfortunately, we usually **can't sample efficiently**. . . .

Approximate Learning with MCMC Marginal Approximation

- An inefficient approach to using an **MCMC approximation of gradient**:
 - ① At iteration k , we want to sample from $p(x | w^k)$.
 - Start from some $x^{k,0}$, sample $x^{k,1}$, sample $x^{k,2}$, etc from an MCMC chain for w^k .
 - Treat the last sample $x^{k,T}$ from the Markov chain as a sample from $p(x | w^k)$.
 - ② Update the parameters using $x^{k,T}$ to get a gradient estimate (sample size 1),

$$w^{k+1} = w^k + \alpha_k (F(\mathbf{X}) - F(x^{k,T})).$$

- If the Markov chain is run long enough, can show convergence using standard stochastic gradient descent arguments.
 - But **have to run MCMC on each iteration** of the SGD method.

Younes Algorithm (“Persistent Contrastive Divergence”)

- Younes algorithm (also known as “persistent contrastive divergence”):

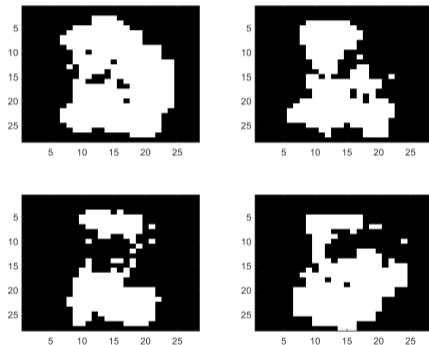
- ① At iteration k , we want to sample from $p(x | w^k)$.
 - Set $x^{k,0} = x^{k-1,T}$, sample $x^{k,1}$, sample $x^{k,2}$, and so on.
 - Treat the last sample $x^{k,T}$ from the Markov chain as a sample from $p(x | w^k)$.
- ② Update the parameters using x^k to get a gradient estimate,

$$w^{k+1} = w^k + \alpha_k (F(\mathbf{X}) - F(x^{k,T})),$$

- In Younes algorithm, you **don't need to run the Markov chain to stationarity**.
 - Usually you only run MCMC for 1 or a small number of iterations.
 - This gives a biased estimate, but is much faster than running MCMC to stationarity.
 - And with small-enough step-size, can show convergence.

Pairwise UGM on MNIST Digits

- Samples from a lattice-structured pairwise UGM trained on MNIST:



- Training: 100k stochastic gradient w/ Gibbs sampling steps with $\alpha_t = 0.01$.
- Samples are iteration 100k of Gibbs sampling with fixed w .

Outline

- 1 Log-Linear Models
- 2 Conditional Random Fields**
- 3 Mixture of Gaussians

Motivation: Rain Data with Month Information

- Our Ising UGM model for the rain data with tied parameters was

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

we switched variable names from x_j to y_c (but model is same).

- First term will reflect that “not rain” is more likely.
- Second term reflects that consecutive days are more likely to be the same.
 - This model is equivalent to a Markov chain model.
- But the model doesn't know that some months are less rainy.
- We can add features that reflect the month (or other information).
 - Multi-label supervised learning, but modeling dependence in labels y_c .
 - Adding fixed features to a UGM is also called a conditional random field (CRF).

Conditional Random Field (CRF) for Rain Data

- A CRF model of rain data, conditioned on 12 “one of k ” month features x_j ,

$$p(y_1, y_2, \dots, y_k \mid \mathbf{x}, w_0, w, v) \propto \exp \left(\sum_{c=1}^k w_0 y_c + \sum_{c=2}^k v y_c y_{c-1} + \sum_{c=1}^k y_c w^\top \mathbf{x} \right).$$

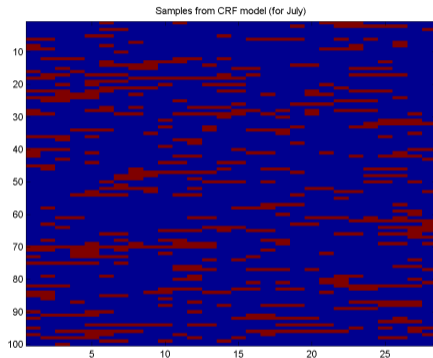
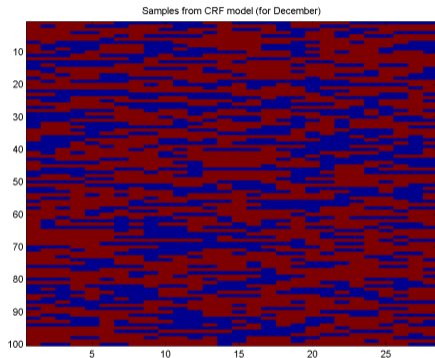
- The potentials in this model over the random variables y_c are

$$\phi_i(y_i) = \exp \left(w_0 y_i + y_i w^\top \mathbf{x} \right), \quad \phi_{ij}(y_i, y_j) = \exp(v y_i y_j).$$

- If we draw the UGM over y_c variables we get a chain structure.
 - So inference can be done using **forward-backward**.
 - And it's still log-linear so the **NLL will be convex**.
 - Gradient descent finds global optimum jointly with respect to w_0 , w , and v .

Rain Data with Month Information

- Samples from CRF conditioned on x being December (left) and July (right):



- Conditional NLL is 16.21, compared to Markov chain which gets NLL 16.81.
 - Mark has Matlab (:/) code for this and a variety of other UGM models:
<https://www.cs.ubc.ca/~schmidtm/Software/UGM.html>

Conditional Random Fields (General Case)

- We often write the likelihood for general CRFs in the form

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

for some parameters w and features $F(\mathbf{x}, y)$.

- The NLL is convex; for a single (\mathbf{x}, y) it's

$$-\log p(y \mid \mathbf{x}, w) = -w^\top F(\mathbf{x}, y) + \log Z(\mathbf{x}, w),$$

with gradient

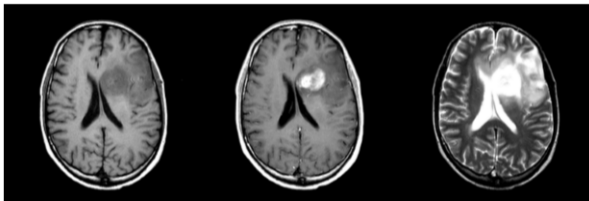
$$-\nabla \log p(y \mid \mathbf{x}, w) = -F(\mathbf{x}, y) + \mathbb{E}_{y \mid \mathbf{x}, w}[F(\mathbf{x}, y)].$$

This requires inference for each value of \mathbf{x} in training data.

- For rain data, need to do run forward-backward 12 times.
- If each example has its own features, need to run it n times.
- Can make sense to use stochastic gradient if n is large.

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.

$$p(y_c | x_c) = \frac{1}{1 + \exp(-2y_c w^\top x_c)} = \frac{\exp(y_c w^\top x_c)}{\exp(w^\top x_c) + \exp(-w^\top x_c)}$$

- 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, **conditional distribution over all labels** in one image is

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

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

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

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

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

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

- Now we have the same “good” logistic regression model, but v **controls how strongly we want neighbours to be the same**.
- We can run gradient descent to **jointly optimize** w and v (convex NLL).
 - So we 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”:
 - Trained with pseudo-likelihood, constraining $v \geq 0$.
 - Decoding with “graph cuts” (see bonus slides from last lecture).



(We were using [edge features](#) $x_{cc'}$ too, see bonus (and different λ on edges).)

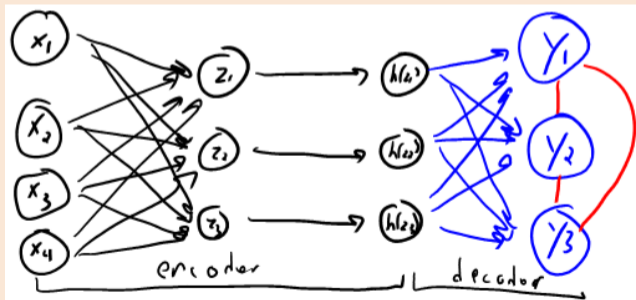
Combining Neural Networks and UGMs

bonus!

- Instead of fixed features, you could use a **neural network**:

$$p(y | x) \propto \exp \left(\sum_{c=1}^k y_c v^T h(W^3 h(W^2 (W^1 x_c))) + \sum_{(c,c') \in \mathcal{E}} u y_c y_{c'} \right).$$

or you could have an encode-decode model spit out potentials of a UGM:



- These are sometimes called a **conditional neural fields** or **deep structured model**.

Multi-Label Classification

bonus!

- Learned dependencies on a multi-label image classification dataset:

female	0.00	0.68	0.04	0.06	0.02	0.24	0.03	-0.00	-0.01	0.01	0.04	-0.00	-0.05	-0.01	0.07	-0.01	-0.00	-0.12	0.04	0.01	0.01	0.02	0.04	0.02
people	0.68	0.00	0.06	0.06	-0.00	0.36	0.03	-0.08	-0.05	-0.03	0.02	-0.06	-0.12	-0.05	0.74	-0.04	-0.03	-0.21	0.01	-0.03	-0.03	-0.03	0.05	-0.03
indoor	0.04	0.06	0.00	0.05	-0.06	0.07	-0.12	-0.07	-0.35	-0.03	-0.46	-0.02	-0.34	0.11	0.02	-0.15	-0.14	-0.01	-0.07	-0.21	0.03	-0.08	0.06	-0.03
baby	0.06	0.06	0.05	0.00	0.10	0.11	0.07	0.09	0.03	0.10	0.01	0.10	0.02	0.09	0.06	0.08	0.07	0.07	0.08	0.06	0.09	0.09	0.08	0.10
sea	0.02	-0.00	-0.06	0.10	0.00	0.04	0.08	0.05	0.16	0.17	-0.02	0.09	-0.02	0.06	0.03	0.14	0.36	0.06	0.05	0.01	0.08	0.14	0.06	0.10
portrait	0.24	0.36	0.07	0.11	0.04	0.00	0.01	0.03	-0.02	0.05	-0.02	0.04	-0.01	0.03	0.12	0.02	0.01	-0.07	0.05	0.05	0.03	0.04	0.07	0.05
transport	0.03	0.03	-0.12	0.07	0.08	0.01	0.00	0.02	0.14	0.07	0.14	0.04	0.05	0.03	0.06	0.08	0.07	-0.03	0.36	0.10	0.04	0.05	0.04	0.07
flower	-0.00	-0.08	-0.07	0.09	0.05	0.03	0.02	0.00	0.02	0.07	-0.03	0.07	0.34	0.04	-0.04	0.04	0.04	0.02	0.05	0.06	0.06	0.06	0.02	0.07
sky	-0.01	-0.05	-0.35	0.03	0.16	-0.02	0.14	0.02	0.00	0.12	0.22	0.04	0.24	-0.02	-0.00	0.44	0.12	-0.04	0.10	0.30	0.01	0.23	0.05	0.11
lake	0.01	-0.03	-0.03	0.10	0.17	0.05	0.07	0.07	0.12	0.00	-0.00	0.09	0.09	0.07	0.01	0.12	0.26	0.06	0.06	0.10	0.07	0.12	0.07	0.18
structures	0.04	0.02	-0.46	0.01	-0.02	-0.02	0.14	-0.03	0.22	-0.00	0.00	0.01	0.04	-0.05	0.06	0.08	-0.04	-0.06	0.14	0.09	-0.00	0.06	0.03	0.02
bird	-0.00	-0.06	-0.02	0.10	0.09	0.04	0.04	0.07	0.04	0.09	0.01	0.00	0.04	0.07	-0.01	0.06	0.09	0.26	0.06	0.05	0.07	0.09	0.05	0.09
plant life	-0.05	-0.12	-0.34	0.02	-0.02	-0.01	0.05	0.34	0.24	0.09	0.04	0.04	0.00	-0.03	-0.07	0.09	0.01	0.01	0.08	0.68	0.02	0.05	-0.07	0.10
food	-0.01	-0.05	0.11	0.09	0.06	0.03	0.03	0.04	-0.02	0.07	-0.05	0.07	-0.03	0.00	-0.01	0.03	0.03	0.03	0.05	0.01	0.06	0.06	0.04	0.07
male	0.07	0.74	0.02	0.06	0.03	0.12	0.06	-0.04	-0.00	0.01	0.06	-0.01	-0.07	-0.01	0.00	0.00	-0.01	-0.10	0.04	-0.02	0.01	0.00	0.06	0.01
clouds	-0.01	-0.04	-0.15	0.08	0.14	0.02	0.08	0.04	0.44	0.12	0.08	0.06	0.09	0.03	0.00	0.00	0.09	-0.00	0.07	0.11	0.05	0.22	-0.01	0.10
water	-0.00	-0.03	-0.14	0.07	0.36	0.01	0.07	0.04	0.12	0.26	-0.04	0.09	0.01	0.03	-0.01	0.09	0.00	0.05	0.02	0.03	0.05	0.10	0.03	0.27
animals	-0.12	-0.21	-0.01	0.07	0.06	-0.07	-0.03	0.02	-0.04	0.06	-0.06	0.26	0.01	0.03	-0.10	-0.00	0.05	0.00	0.02	0.00	0.22	0.03	-0.01	0.05
car	0.04	0.01	-0.07	0.08	0.05	0.05	0.36	0.05	0.10	0.06	0.14	0.06	0.08	0.05	0.04	0.07	0.02	0.02	0.00	0.11	0.06	0.08	0.07	0.06
tree	0.01	-0.03	-0.21	0.06	0.01	0.05	0.10	0.06	0.30	0.10	0.09	0.05	0.68	0.01	-0.02	0.11	0.03	0.00	0.11	0.00	0.04	0.09	-0.00	0.12
dog	0.01	-0.03	0.03	0.09	0.08	0.03	0.04	0.06	0.01	0.07	-0.00	0.07	0.02	0.06	0.01	0.05	0.05	0.22	0.06	0.04	0.00	0.06	0.05	0.07
sunset	0.02	-0.03	-0.08	0.09	0.14	0.04	0.05	0.06	0.23	0.12	0.06	0.09	0.05	0.06	0.00	0.22	0.10	0.03	0.08	0.09	0.06	0.00	0.06	0.10
night	0.04	0.05	0.06	0.08	0.06	0.07	0.04	0.02	0.05	0.07	0.03	0.05	-0.07	0.04	0.06	-0.01	0.03	-0.01	0.07	-0.00	0.05	0.06	0.00	0.07
river	0.02	-0.03	-0.03	0.10	0.10	0.05	0.07	0.07	0.11	0.18	0.02	0.09	0.10	0.07	0.01	0.10	0.27	0.05	0.06	0.12	0.07	0.10	0.07	0.00

Automatic Differentiation (AD) vs. Inference

bonus!

- Deep structured model gradient combines neural/Markov gradients:
 - ① **Forward pass** through neural network to get \hat{y}_c predictions.
 - ② **Forward message passing** to compute normalizing constant.
 - ③ **Backwards message passing** to compute marginals.
 - ④ **Backwards pass** through neural network to get all gradients.
- You could skip the last two steps if you use **automatic differentiation**.
- But with **approximate** inference, AD may or may not work:
 - AD will **work for iterative variational inference** methods (which we'll cover later).
 - But it takes **way more memory** than needed (needs to store all iterations).
 - AD is **harder for Monte Carlo methods**.
 - Can't AD through sampling steps – but can use “reparameterization trick” (later).
- Recent trend: run **iterative variational method for a fixed number of iterations**.
 - AD can give gradient of result after this fixed number of iterations.
 - “Train the inference you will use at test time.”

Combining FCNs and CRFs

bonus!

- DeepLab used a fully-connected pairwise UGM on top layer of FCN:

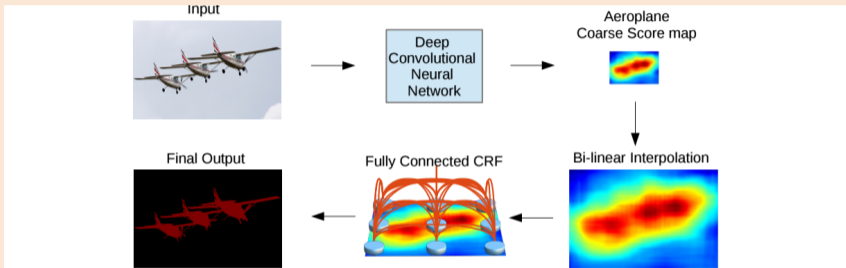


Fig. 1: Model Illustration. A Deep Convolutional Neural Network such as VGG-16 or ResNet-101 is employed in a fully convolutional fashion, using atrous convolution to reduce the degree of signal downsampling (from 32x down 8x). A bilinear interpolation stage enlarges the feature maps to the original image resolution. A fully connected CRF is then applied to refine the segmentation result and better capture the object boundaries.

<https://arxiv.org/pdf/1606.00915.pdf>

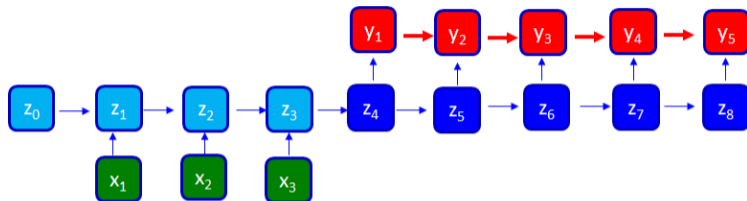
- Most recent version of the paper **removed the UGM**.
- Still really helps if you don't have tons of training data (Bae, . . . , Sutherland, IJCAI-23).

Do we need UGMs in Neural Networks?

- Recall that **encode-decode hidden layers already capture label dependencies**.
 - So do we need a UGM to explicitly model label dependencies in output layer?
- Factor 1: data size (big vs. small).
 - With a **small dataset**, it could be helpful to have direct dependencies in model.
 - With a **large dataset**, the hidden layers should reflect dependencies.
- Factor 2: how you evaluate the model (individual parts or full decoding).
 - If you measure “pixel level” or “word level” error, UGMs may not help.
 - If you measure “whole image” or “whole sentence” error, UGMs may help.
 - Because for example inference can discourage unlikely joint labelings.

Combining RNNs and Graphical Models

- An example where we use **explicit label dependencies** is language translation:

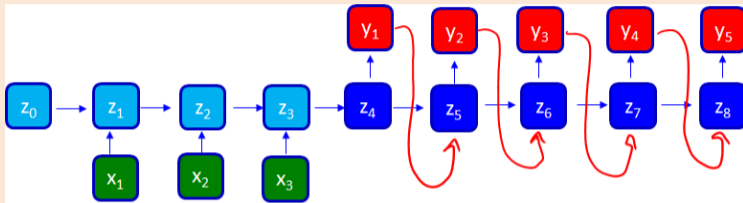


- Above model has usual deterministic edges, and **DAG edges** on labels.
- Can use **Viterbi decoding to find best translation** in this model.
 - Taking into account probability of seeing neighbouring words.
- But there is not much information in the DAG part of the model.
 - Only modeling dependencies between adjacent words.
- What we really want is to have the **label we output affect the hidden state**.
 - So that the **encoding reflects previously-output words**.

Combining RNNs and Graphical Models

bonus!

- In order for the hidden states to depend on the output, we have this monstrosity:



- This can still be written as a Markov chain, but we **cannot do Viterbi decoding**.
 - Problem is that the **hidden states in decoder become random variables**.
 - So the state at each time has discrete and continuous parts (cannot be enumerated).
- To do decoding in this thing, we typically use **beam search**.
 - Heuristic algorithm that maintains “ k best decodings up to time t .”
 - Can be arbitrarily bad, but works if decoding is obvious as we go forward in time.
 - The type of edge and decoding strategy is also common with **transformers**.

Summary

- **Log-linear** parameterization can be used to learn UGMs:
 - Maximum likelihood is convex, but requires normalizing constant and inference.
- **Approximate UGM learning:**
 - ① Change objective function: **pseudolikelihood**.
 - ② Approximate marginals: Monte Carlo or variational methods.
 - **Younes algorithm** for using MCMC within SGD.
- **Conditional random fields** generalize logistic regression:
 - Multi-label model that explicitly models label dependencies.
- **Combining CRFs with deep learning.**
 - You can learn features and the explicit label dependencies.

End of Part 4 (“Markov Models”): Key Concepts

- We discussed **Markov chains**:
 - Distribution assuming independence of past given last time (**Markov assumption**).
 - Common parameterization uses **initial probabilities** and **transition probabilities**.
 - **Homogeneous** Markov chains assume **same transition probabilities** across time.
- We discussed **inference in Markov chains**.
 - **Ancestral sampling**: sample each variable given previous variables in ordering.
 - **CK equations**: give marginals recursively.
 - **Stationary distribution**: marginals as time goes to infinity.
 - **Viterbi decoding**: special case of dynamic programming.
 - **Forward backward**: computation of all conditionals with two “passes”.

End of Part 4 (“Markov Models”): Key Concepts

- We discussed **Markov chain Monte Carlo (MCMC)**:
 - Define a Markov chain that has target distribution as stationary distribution.
 - Use samples from the Markov chain within Monte Carlo method.
 - Possibly with **burn in** and/or **thinning**.
 - Most common methods are **Metropolis-Hastings**.
 - Based on accepting proposals or keeping the same sample.
 - Special case of Metropolis-Hastings is **Gibbs sampling**.
 - Based on sampling one variable at a time given all others.

End of Part 4 (“Markov Models”): Key Concepts

- We discussed **directed acyclic graphical (DAG)**.
 - Assume independence of previous variables given a set of **parent** variables.
 - Can be used to visualize models/assumptions.
 - Conditional independences can be tested using **d-separation**.
 - Are paths blocked by observed chain/fork, or unobserved child?
 - Our **standard independence assumptions** appear if we add parameters to DAG.
 - Training DAGs decomposes into d supervised learning problems.
- We discussed **undirected graphical models (UGMs)**.
 - Write distribution as product of non-negative **potentials** over subsets of variables.
 - **Log-linear** models use $\exp(\text{linear})$ potentials.
 - Convex NLL trained with gradient descent, but gradient **requires inference**.
 - Approximate training methods include pseudo-likelihood and variational methods.
 - Or Younes algorithm which integrates SGD steps within MCMC.
 - **Conditional random fields** add features to UGMs.
 - **Deep structured models** learn features in UGMs.

End of Part 4 (“Markov Models”): Key Concepts

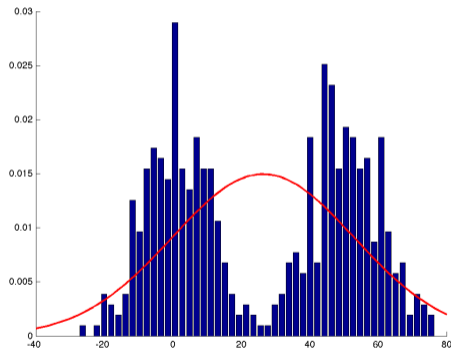
- We briefly discussed **inference in graphical models**.
 - Markov chain inference methods extend to trees for DAGs and UGMs.
 - But for general graphs inference can be hard in DAGs/UGMs.
 - Except unconditional sampling, likelihood, and learning (easy in DAGs).
- We skipped over **structured SVMs**
 - A generalization of SVMs that can model correlations in labels.
 - Applying SGD requires **decoding instead of inference**.
 - Mark’s slides on this topic are here:
<https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/L28.5.pdf>

Outline

- 1 Log-Linear Models
- 2 Conditional Random Fields
- 3 Mixture of Gaussians**

1 Gaussian for Multi-Modal Data

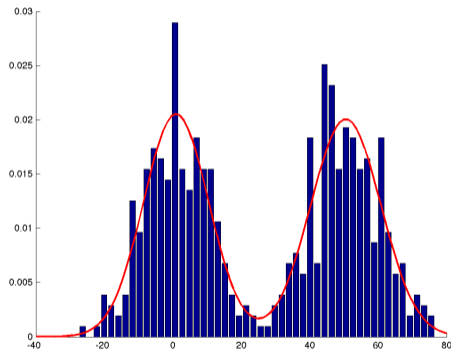
- Major drawback of Gaussian is that it is **uni-modal**.
 - It gives a terrible fit to data like this:



- If Gaussians are all we know, how can we fit this data?

2 Gaussians for Multi-Modal Data

- We can fit this data by using **two Gaussians**



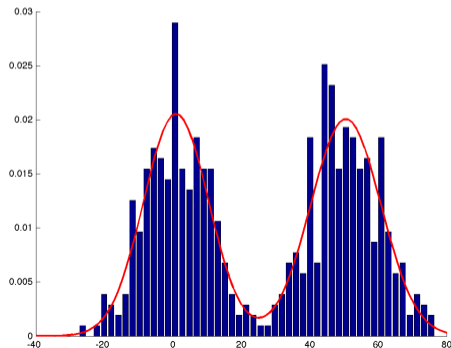
- Half the samples are from Gaussian 1, half are from Gaussian 2.

Mixture of Gaussians

- Our probability density in this example is given by

$$p(x^i | \mu_1, \mu_2, \Sigma_1, \Sigma_2) = \frac{1}{2} \underbrace{p(x^i | \mu_1, \Sigma_1)}_{\text{PDF of Gaussian 1}} + \frac{1}{2} \underbrace{p(x^i | \mu_2, \Sigma_2)}_{\text{PDF of Gaussian 2}},$$

- We need the (1/2) factors so it still integrates to 1.



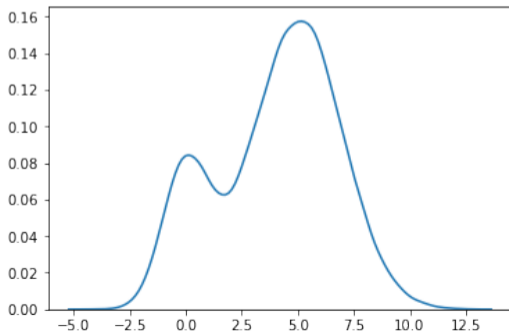
Mixture of Gaussians

- If data comes from **one Gaussian more often** than the other, we could use

$$p(x^i | \mu_1, \mu_2, \Sigma_1, \Sigma_2, \pi_1, \pi_2) = \pi_1 \underbrace{p(x^i | \mu_1, \Sigma_1)}_{\text{PDF of Gaussian 1}} + \pi_2 \underbrace{p(x^i | \mu_2, \Sigma_2)}_{\text{PDF of Gaussian 2}},$$

where π_1 and π_2 are non-negative and sum to 1.

- π_1 represents “probability that we take a sample from Gaussian 1”.

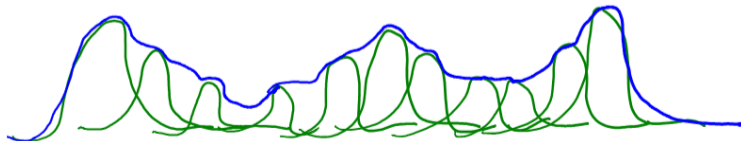


Mixture of Gaussians

- In general we might have a **mixture of k Gaussians** with different weights.

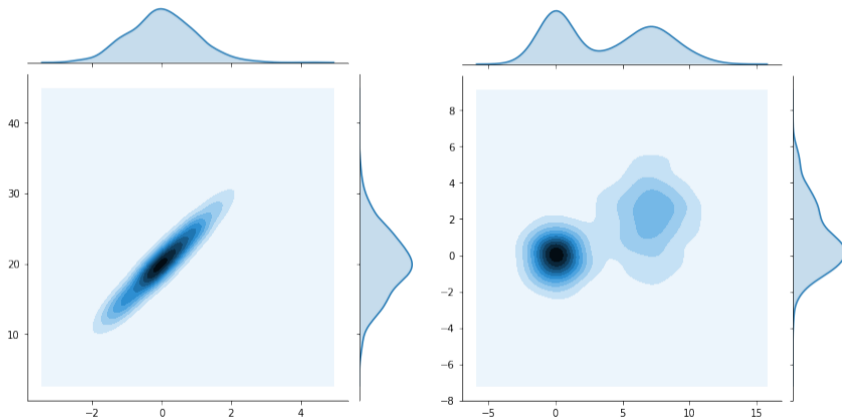
$$p(x | \mu, \Sigma, \pi) = \sum_{c=1}^k \pi_c \underbrace{p(x | \mu_c, \Sigma_c)}_{\text{PDF of Gaussian } c},$$

- Where π_c are categorical distribution parameters (non-negative and sum to 1).
- We can use it to model complicated densities with Gaussians (like RBFs).
 - “Universal approximator”: can model any continuous density on compact set.



Mixture of Gaussians

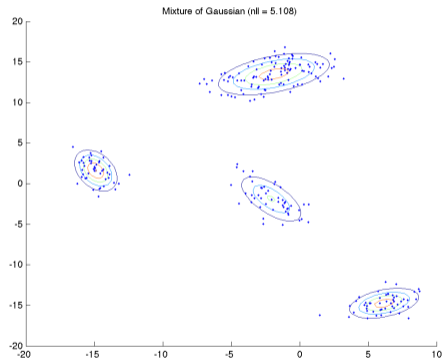
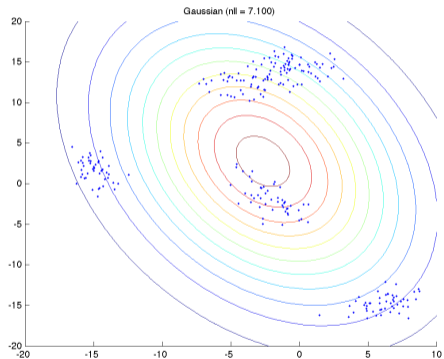
- Gaussian vs. mixture of 2 Gaussian densities in 2D:



- Marginals will also be mixtures of Gaussians.

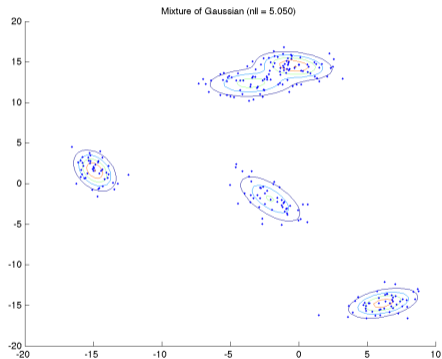
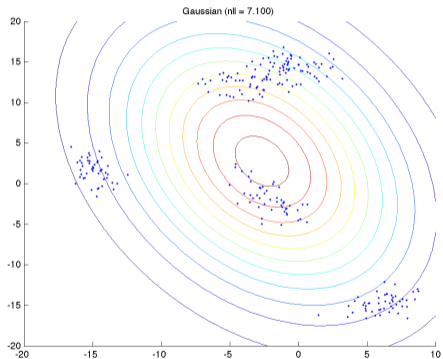
Mixture of Gaussians

- Gaussian vs. Mixture of 4 Gaussians for 2D multi-modal data:



Mixture of Gaussians

- Gaussian vs. Mixture of 5 Gaussians for 2D multi-modal data:



Latent-Variable Representation of Mixtures

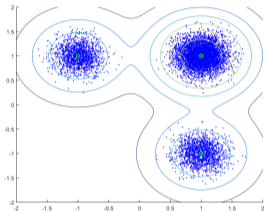
- For inference/learning in mixture models, we often **introduce variables** z^i .
 - Each z^i is a categorical variable in $\{1, 2, \dots, k\}$ when we have k mixtures.
 - The value z^i represents “what mixture this example came from”.
 - We **do not observe the** z^i values (they are called **latent variables**).
- Why this interpretation of “**each** x^i **comes from one Gaussian**”?
 - Consider a model where $p(z^i = c) = \pi_c$, and $x^i | z^i = c \sim \mathcal{N}(\mu_c, \Sigma_c)$.
 - Now **marginalize over the** z^i in this model:

$$\begin{aligned} p(x | \mu, \Sigma, \pi) &= \sum_{c=1}^k p(x, z = c) = \sum_{c=1}^k p(z = c)p(x | z = c) \\ &= \sum_{c=1}^k \pi_c \underbrace{p(x | \mu_c, \Sigma_c)}_{\text{PDF of Gaussian } c}, \end{aligned}$$

which is the **PDF of the mixture** of Gaussians model.

Ancestral Sampling in Mixture of Gaussians

- Generating samples with **ancestral sampling in the latent variable representation**:
 - 1 **Sample cluster z** based on prior probabilities π_c (categorical distribution).
 - 2 **Sample example x** based on mean μ_z and covariance Σ_z of Gaussian z .



- Marginalization and computing conditionals is also easy.
- Decoding z or computing marginal $p(z | x)$ is easy (next slide).
- Decoding x in Gaussian mixtures is NP-hard.
- We usually fit these models with **expectation maximization** (EM).
- Choosing k : domain knowledge, test set likelihood, or marginal likelihood.

Inference Task: Computing Responsibilities

- Consider computing **probability that example i came from mixture c** .
 - We call this the **responsibility** of mixture c for example i ,

$$\begin{aligned}r_c^i &= p(z = c \mid x^i) \\&= \frac{p(z = c, x^i)}{p(x^i)} \\&= \frac{p(z = c, x^i)}{\sum_{c'=1}^k p(z' = c, x^i)} \\&= \frac{p(z = c) p(x^i \mid z = c)}{\sum_{c'=1}^k p(z' = c) p(x^i \mid z' = c)} \\&= \frac{\pi_c p(x^i \mid \mu_c, \Sigma_c)}{\sum_{c'=1}^k \pi_{c'} p(x^i \mid \mu_{c'}, \Sigma_{c'})} \quad (\text{we know all these values})\end{aligned}$$

- If you think the different mixtures as clusters, this is **probability of being in cluster**.

Notation Alert: π vs. z vs. r (MEMORIZE)

- In mixture models, many people **confuse the quantities π , z , and r** .
 - Vector π has k elements in $[0, 1]$ and summing up to 1.
 - Number π_c is the “prior” probability that an example is in cluster c .
 - This is a **parameter** (we learn it from data).
 - Matrix \mathbf{R} is $n \times k$ matrix, summing to 1 across rows.
 - Number r_c^i is the “posterior” probability that example i is in cluster c .
 - Computing these values is an **inference task** (assumes known parameters).
 - Vector z has n elements in $\{1, 2, \dots, k\}$.
 - Category z^i is the **actual mixture/cluster** that generated example i .
 - This is a **nuisance parameter** (an unknown variable that is not a parameter).

Summary

- **Mixture of Gaussians** writes probability as convex comb. of Gaussian densities.
 - Can model arbitrary continuous densities.
- **Latent-variable** representation of mixtures with cluster variables z^i .
 - Allows ancestral sampling by sampling cluster than example.
 - **Responsibility** is probability that an example belongs to a cluster.

Example: Ising Model of Rain Data

bonus!

- E.g., for the rain data we could parameterize our node potentials using

$$\log(\phi_i(x_i)) = \begin{cases} w_1 & \text{no rain} \\ 0 & \text{rain} \end{cases} .$$

- Why do we only need 1 parameter?
 - Scaling $\phi_i(1)$ and $\phi_i(2)$ by constant doesn't change distribution.
- In general, we only need $(k - 1)$ parameters for a k -state variable.
 - But if we're using regularization we may want to use k anyways (symmetry).

Example: Ising Model of Rain Data

bonus!

- The **Ising parameterization** of edge potentials,

$$\log(\phi_{ij}(x_i, x_j)) = \begin{cases} w_2 & x_i = x_j \\ 0 & x_i \neq x_j \end{cases}.$$

- Applying gradient descent gives MLE of

$$w = \begin{bmatrix} 0.16 \\ 0.85 \end{bmatrix}, \quad \phi_i = \begin{bmatrix} \exp(w_1) \\ \exp(0) \end{bmatrix} = \begin{bmatrix} 1.17 \\ 1 \end{bmatrix}, \quad \phi_{ij} = \begin{bmatrix} \exp(w_2) & \exp(0) \\ \exp(0) & \exp(w_2) \end{bmatrix} = \begin{bmatrix} 2.34 & 1 \\ 1 & 2.34 \end{bmatrix},$$

preference towards no rain, and **adjacent days being the same**.

- Average NLL of 16.8 vs. 19.0 for independent model.

- We could alternately use fully expressive edge potentials

$$\log(\phi_{ij}(x_i, x_j)) = \begin{bmatrix} w_2 & w_3 \\ w_4 & w_5 \end{bmatrix},$$

but these don't improve the likelihood much.

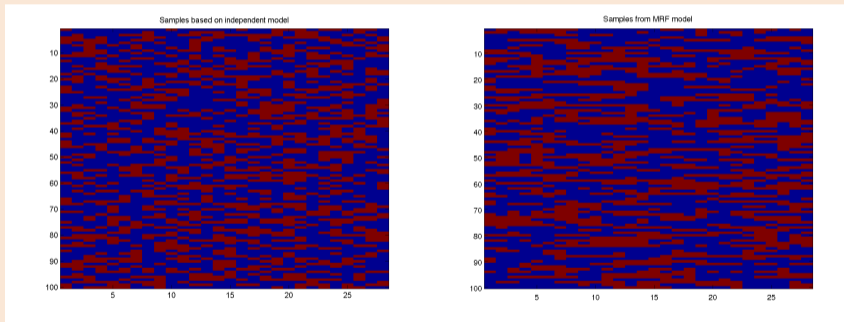
- We could fix one of these at 0 due to the normalization.
 - But we often don't do this when using regularization.
- We could also have special **potentials for the boundaries**.
 - Many language models are homogeneous, except for start/end of sentences.

Example: Ising Model of Rain Data

bonus!

Independent model vs. chain-UGM model with **tied nodes and Ising tied edges**:

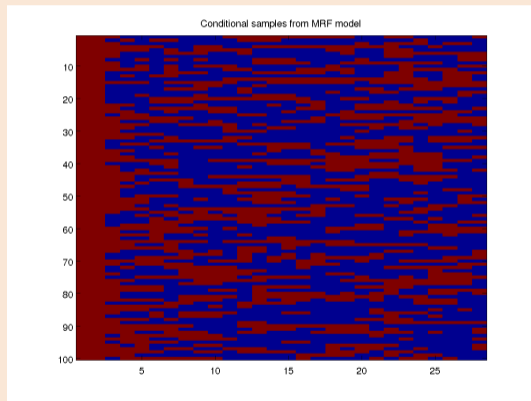
- For this dataset, using untied or general edges doesn't change likelihood much.



Example: Ising Model of Rain Data

bonus!

Samples from Ising chain-UGM model if it rains on the first day:



Example of Feature Function

bonus!

- Consider the 2-node 1-edge UGM (1)–(2), where each state has 2 values.
 - So we have potentials $\phi_1(x_1)$, $\phi_2(x_2)$, and $\phi_{12}(x_1, x_2)$ and want to have

$$w^\top F(x) = w_{1,x_1} + w_{2,x_2} + w_{1,2,x_1,x_2}.$$

- With no parameter tying and $x = [2 \ 1]$, our parameter vector and features are

$$w = \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{2,1} \\ w_{2,2} \\ w_{1,2,1,1} \\ w_{1,2,1,2} \\ w_{1,2,2,1} \\ w_{1,2,2,2} \end{bmatrix}, \quad F(x) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

Example of Feature Function

bonus!

- If we instead had Ising potentials (just measuring whether $x_1 = x_2$) we would have

$$w^T F(x) = w_{1,x_1} + w_{2,x_2} + w_{1,2,\text{same}},$$

where $w_{1,2,\text{same}}$ is the parameter specifying how much we want $x_1 = x_2$.

- With no parameter tying and $x = [2 \ 1]$, our parameter vector and features are

$$w = \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{2,1} \\ w_{2,2} \\ w_{1,2,\text{same}} \end{bmatrix}, \quad F(x) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

UGM Training Objective Function

bonus!

- With log-linear parameterization, NLL for IID training examples is

$$\begin{aligned} f(w) &= -\sum_{i=1}^n \log p(x^i | w) = -\sum_{i=1}^n \log \left(\frac{\exp(w^\top F(x^i))}{Z(w)} \right) \\ &= -\sum_{i=1}^n w^\top F(x^i) + \sum_{i=1}^n \log Z(w) \\ &= -w^\top F(\mathbf{X}) + n \log Z(w). \end{aligned}$$

where the $F(\mathbf{X}) = \sum_i F(x^i)$ are called the **sufficient statistics** of the dataset.

- Given sufficient statistics $F(\mathbf{X})$, we can throw out the examples x^i .
(only go through data once)
- Function $f(w)$ is **convex** (it's linear plus a big log-sum-exp function).
 - But notice that Z depends on w

- For 1 example x , we showed that NLL with log-linear parameterization is

$$f(w) = -w^T F(\mathbf{X}) + \log Z(w).$$

- The partial derivative with respect to parameter w_j has a simple form

$$\begin{aligned}\nabla_{w_j} f(w) &= -F_j(\mathbf{X}) + \sum_x \frac{\exp(w^T F(x))}{Z(w)} F_j(x) \\ &= -F_j(\mathbf{X}) + \sum_x p(x | w) F_j(x) \\ &= -F_j(\mathbf{X}) + \mathbb{E}[F_j(x)].\end{aligned}$$

- Observe that **derivative of $\log(Z)$ is expected value of feature.**

Brain Tumour Segmentation with Label Dependencies

bonus!

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

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

- 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


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

so it didn't make inference any more complicated.

Modeling OCR Dependencies

bonus!

- What dependencies should we model for this problem?

Input: 

Output: "Paris"

- $\phi(y_c, x_c)$: 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.
- $\phi_c(y_{c-1}, y_c)$: inhomogeneous dependency (French: 'e-r' ending).
- $\phi_c(y_{c-2}, y_{c-1}, y_c)$: third-order and inhomogeneous (English: 'i-n-g' end).
- $\phi(y \in \mathcal{D})$: is y in dictionary \mathcal{D} ?

Tractability of Discriminative Models

bonus!

- 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 tasks 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).
 - For example, in rain data the seasons will be approximately 3 months.
 - **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 (eventually better but probably much slower).
 - ③ Variational approximate inference (fast, quality varies).

Structure Learning in UGMs

bonus!

- Recall that in **Ising** UGMs, our edge potentials have the form

$$\phi_{ij}(x_i, x_j) = \exp(w_{ij}x_i x_j).$$

- If we set $w_{ij} = 0$, it sets $\phi_{ij}(x_i, x_j) = 1$ for all x_i and x_j .
 - Potential just “multiplies by 1”, which is **equivalent to removing the edge**.
- **L1-regularization of w_{ij}** values performs **structure learning in UGM**.
- For general log-linear, each **edge has multiple parameters** $w_{i,j,s,s'}$.
 - In this case we can use “**group L1-regularization**” for structure learning.
 - Each group will be all parameters $w_{i,j,\cdot,\cdot}$ associated with an edge (i, j) .

Structure Learning on Rain Data

bonus!

Large λ (and optimal tree):



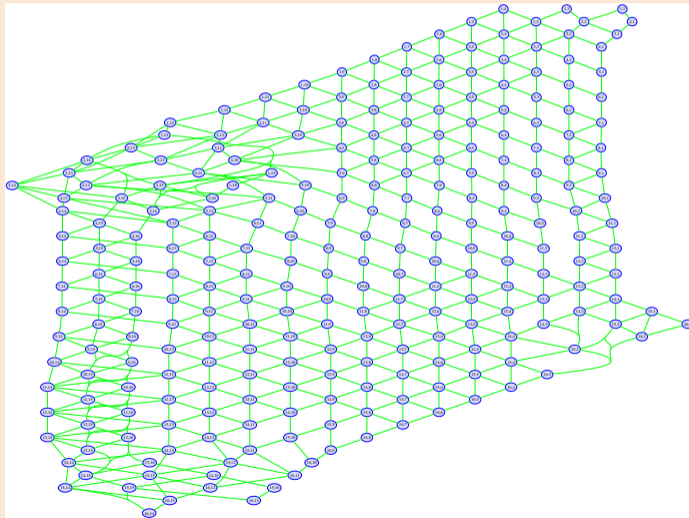
Small λ :



Structure Learning on USPS Digits

bonus!

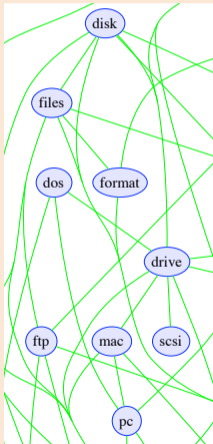
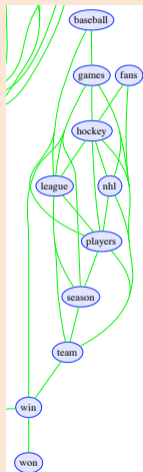
Structure learning of pairwise UGM with group-L1 on USPS digits:



Structure Learning on News Words

bonus!

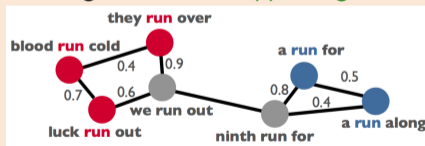
Group-L1 on newsgroups data:



Posterior Regularization

bonus!

- In some cases it might make sense to use **posterior regularization**:
 - Regularize the probabilities in the resulting model.
- Consider an NLP labeling task where
 - You have a small amount of labeled sentences.
 - You have a huge amount of unlabeled sentences.
- Maximize labeled likelihood, plus **total-variation penalty on $p(y_c | x, w)$ values**.
 - Give high regularization weights to **words appearing in same trigrams**:



<http://jgillenw.com/conll2013-talk.pdf>

- Useful for “out of vocabulary” words (words that don’t appear in labeled data).
 - Has been replaced in recent by continuous word representations like word2vec.

Avoiding Underflow when Computing Responsibilities

bonus!

- Computing responsibility may underflow for high-dimensional x^i , due to $p(x^i | z^i = c, \Theta^t)$.
- Usual ML solution: do all but last step in log-domain.

$$\begin{aligned} \log r_c^i &= \log p(x^i | z^i = c, \Theta^t) + \log p(z^i = c | \Theta^t) \\ &\quad - \log \left(\sum_{c'=1}^k p(x^i | z^i = c', \Theta^t) p(z^i = c' | \Theta^t) \right). \end{aligned}$$

- To compute **last** term, use “log-sum-exp” trick.

Log-Sum-Exp Trick

bonus!

- To compute $\log(\sum_i \exp(v_i))$, set $\beta = \max_i \{v_i\}$ and use:

$$\begin{aligned}\log\left(\sum_c \exp(v_i)\right) &= \log\left(\sum_i \exp(v_i - \beta + \beta)\right) \\ &= \log\left(\sum_i \exp(v_i - \beta) \exp(\beta)\right) \\ &= \log(\exp(\beta)) \sum_i \exp(v_i - \beta) \\ &= \log(\exp(\beta)) + \log\left(\sum_i \exp(v_i - \beta)\right) \\ &= \beta + \log\left(\sum_i \underbrace{\exp(v_i - \beta)}_{\leq 1}\right).\end{aligned}$$

- Avoids overflows due to computing exp operator.