Binary Classification and Feature Selection via Generalized Approximate Message Passing

Phil Schniter

THE OHIO STATE UNIVERSITY

Collaborators: Justin Ziniel (OSU-ECE) and Per Sederberg (OSU-Psych)

Supported in part by NSF grant CCF-1018368, NSF grant CCF-1218754 and DARPA/ONR grant N66001-10-1-4090.

CISS (Princeton) — March'14

Binary Linear Classification

- Observe m training examples $\{(y_i, \boldsymbol{a}_i)\}_{i=1}^m$, each comprised of a binary label $y_i \in \{-1, 1\}$ and a feature vector $\boldsymbol{a}_i \in \mathbb{R}^n$.

- Assume that data follows a generalized linear model:

$$\Pr\{y_i = 1 \,|\, \boldsymbol{a}_i; \boldsymbol{x}_{\text{true}}\} = p_{Y|Z}(1 \,|\, \underbrace{\boldsymbol{a}_i^{\mathsf{T}} \boldsymbol{x}_{\text{true}}}_{\triangleq z_{i,\text{true}}})$$

  for some "true" weight vector $\boldsymbol{x}_{\text{true}} \in \mathbb{R}^n$ and some activation function $p_{Y|Z}(1 \,|\, \cdot) : \mathbb{R} \to [0, 1]$.

- Goal 1: estimate $\hat{\boldsymbol{x}}_{\text{train}} \approx \boldsymbol{x}_{\text{true}}$ from training data, so to be able to predict the unknown label y_{test} associated with a test vector $\boldsymbol{a}_{\text{test}}$:

$$\text{compute } \Pr\{y_{\text{test}} = 1 \,|\, \boldsymbol{a}_{\text{test}}; \hat{\boldsymbol{x}}_{\text{train}}\} = p_{Y|Z}(1 \,|\, \boldsymbol{a}_{\text{test}}^{\mathsf{T}} \hat{\boldsymbol{x}}_{\text{train}})$$

Binary Feature Selection

- Operating regimes:
  - $m \gg n$: Plenty of training examples: feasible to learn $\hat{\boldsymbol{x}}_{\text{train}} \approx \boldsymbol{x}_{\text{true}}$.
  - $m \ll n$: Training-starved: feasible if $\boldsymbol{x}_{\text{true}}$ is sufficiently sparse!

- The training-starved case motivates...

  Goal 2: Identify salient features (i.e., recover support of $\boldsymbol{x}_{\text{true}}$).

  - Example: From fMRI, learn which parts of the brain are responsible for discriminating two classes of object (e.g., cats vs. houses):

$$\begin{aligned} n = 31398 \quad &\leftrightarrow \quad \text{fMRI voxels} \\ m = 216 \quad &\leftrightarrow \quad 2 \text{ classes} \times 9 \text{ examples} \times 12 \text{ subjects} \end{aligned}$$

  - Can interpret as support recovery in noisy one-bit compressed sensing:

$$\boldsymbol{y} = \operatorname{sgn}(\boldsymbol{A}\boldsymbol{x}_{\text{true}} + \boldsymbol{w}) \text{ with i.i.d noise } \boldsymbol{w}.$$

Bring out the GAMP

Zed: Bring out the Gimp.

Maynard: Gimp's sleeping.

Zed: Well, I guess you're gonna have to go wake him up now, won't you?

—Pulp Fiction, 1994.

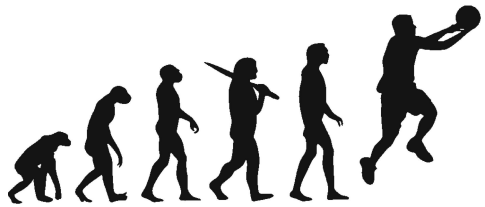
We propose a new approach to binary linear classification and feature selection based on generalized approximate message passing (GAMP).

Advantages of GAMP include

- flexibility in choosing activation $p_{Y|Z}$ & weight prior p_X
- excellent accuracy & runtime
- state-evolution governing behavior in large-system limit
- can tune without cross-validation (via EM extension [Vila & S. '11])
- can learn & exploit structured sparsity (via turbo extension [S. '10])

Approximate Message Passing

- AMP is derived from a simplification of message passing (sum-product or max-sum) that holds in the large-system limit.
- AMP manifests as a sophisticated form of iterative thresholding, requiring only two applications of $\boldsymbol{A}$ per iteration and few iterations.



- The evolution of AMP:
  - The original AMP [Donoho, Maleki, Montanari '09] solved the LASSO problem $\arg\min_{\boldsymbol{x}} \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\lambda\|\boldsymbol{x}\|_1$ assuming i.i.d sub-Gaussian $\boldsymbol{A}$.
  - The Bayesian AMP [Donoho, Maleki, Montanari '10] extended to MMSE inference in AWGN for any factorizable signal prior $\prod_j p_X(x_j)$.
  - The generalized AMP [Rangan '10] framework extends to MAP or MMSE inference under any factorizable signal prior & likelihood.

GAMP Theory

- In the large-system limit with i.i.d sub-Gaussian $\boldsymbol{A}$, GAMP follows a state-evolution trajectory whose fixed points are MAP/MMSE optimal solutions when unique [Rangan '10], [Javanmard, Montanari '12]
- With arbitrary finite-dimensional $\boldsymbol{A}$,
  - the fixed-points of max-product GAMP coincide with the critical points of the MAP optimization objective

$$\arg\max_{\boldsymbol{x}} \left\{ \sum_{i=1}^{m} \log p_{Y_i|Z_i}(y_i|[\boldsymbol{A}\boldsymbol{x}]_i) + \sum_{j=1}^{n} \log p_{X_j}(x_j) \right\}$$

  - the fixed-points of sum-product GAMP coincide with the critical points of a certain free-energy optimization objective [Rangan, S., et al'13]

and damping can be used to ensure that GAMP converges to its fixed points. [Rangan,S.,Fletcher'14]

GAMP for Binary Classification and Feature Selection

So, how do we use GAMP to design the weight vector $\hat{\boldsymbol{x}}$?

1. Choose GAMP's linear transform $\boldsymbol{A}$:
   - Linear classification: the rows of $\boldsymbol{A}$ are the feature vectors $\{\boldsymbol{a}_i^{\mathsf{T}}\}_{\forall i}$.
   - Kernel-based classification: $[\boldsymbol{A}]_{i,j} = \mathcal{K}(\boldsymbol{a}_i, \boldsymbol{a}_j)$ with appropriate $\mathcal{K}(\cdot,\cdot)$.
2. Choose inference mode:
   - max-sum: finds $\hat{\boldsymbol{x}}$ that minimizes regularized loss, i.e.,

     $$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \left\{ \sum_{i=1}^m f([\boldsymbol{A}\boldsymbol{x}]_i; y_i) + \sum_{j=1}^n g(x_j) \right\} \text{ for chosen } f \text{ and } g$$

   - sum-product: computes the marginal weight posteriors $p_{X_j|\boldsymbol{Y}}(\cdot|\boldsymbol{y})$ under the assumed statistical model:

     $$\Pr\{\boldsymbol{y}|\boldsymbol{A},\boldsymbol{x}\} = \textstyle\prod_{i=1}^m p_{Y|Z}(y_i|\boldsymbol{a}_i^{\mathsf{T}}\boldsymbol{x}) \quad \text{and} \quad p(\boldsymbol{x}) = \textstyle\prod_{j=1}^n p_X(x_j).$$

3. Choose activation fxn $p_{Y|Z}(y_i|\cdot) \propto e^{-f(\cdot;y_i)}$ and prior $p_X(\cdot) \propto e^{-g(\cdot)}$

GAMPmatlab: Implemented Activations and Priors

For given $p_{Y|Z}$ and p_X, GAMP needs to compute the mean and variance (sum-product), or the max and sensitivity (max-sum), of the scalar pdfs

$$p_{Z|Y}(z|y;\mu_Q,v_Q) \propto p_{Y|Z}(y|z)\mathcal{N}(z;\mu_Q,v_Q)$$
$$p_{X|Q}(x|q;\mu_R,v_R) \propto p_X(x)\mathcal{N}(x;\mu_R,v_R)$$

Our `http://sourceforge.net/projects/gampmatlab/` implementation handles these computations for various common choices of $p_{Y|Z}$ and p_X:

activation: $p_{Y\|Z}$	sum-prod	max-prod
logit	VI	RF
probit	CF	RF
hinge	CF	RF
robust-*	CF	RF

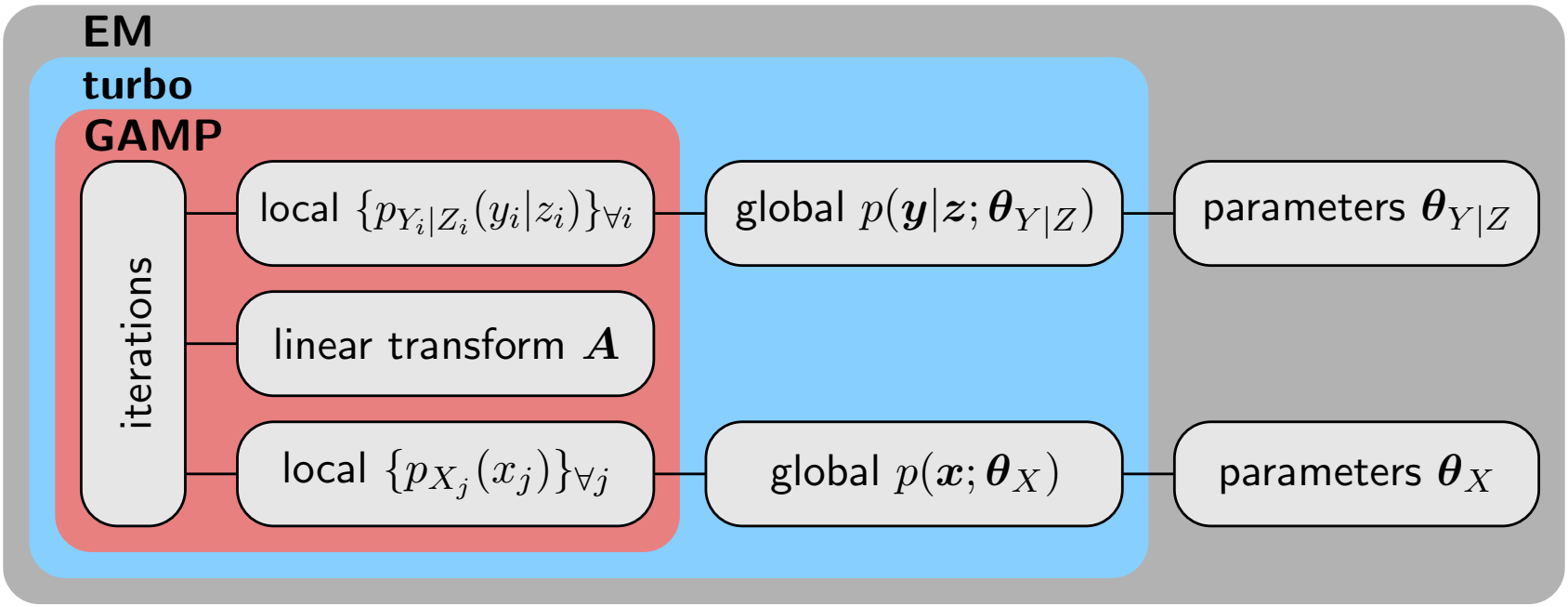
prior: p_X	sum-prod	max-prod
Gaussian	CF	CF
Laplace	CF	CF
Elastic Net	CF	CF
Bernoulli-*	CF	–

CF=closed-form, NI=numerical integration, VI=variational inference, RF=root-finding.

Beyond GAMP: The EM & turbo Extensions

- The basic GAMP algorithm requires
  1. separable priors $p(\boldsymbol{y}|\boldsymbol{z}) = \prod_i p_{Y_i|Z_i}(y_i|z_i)$ and $p(\boldsymbol{x}) = \prod_j p_{X_j}(x_j)$
  2. that are perfectly known.

- The EM-turbo-GAMP framework circumvents these limitations by learning possibly non-separable priors:



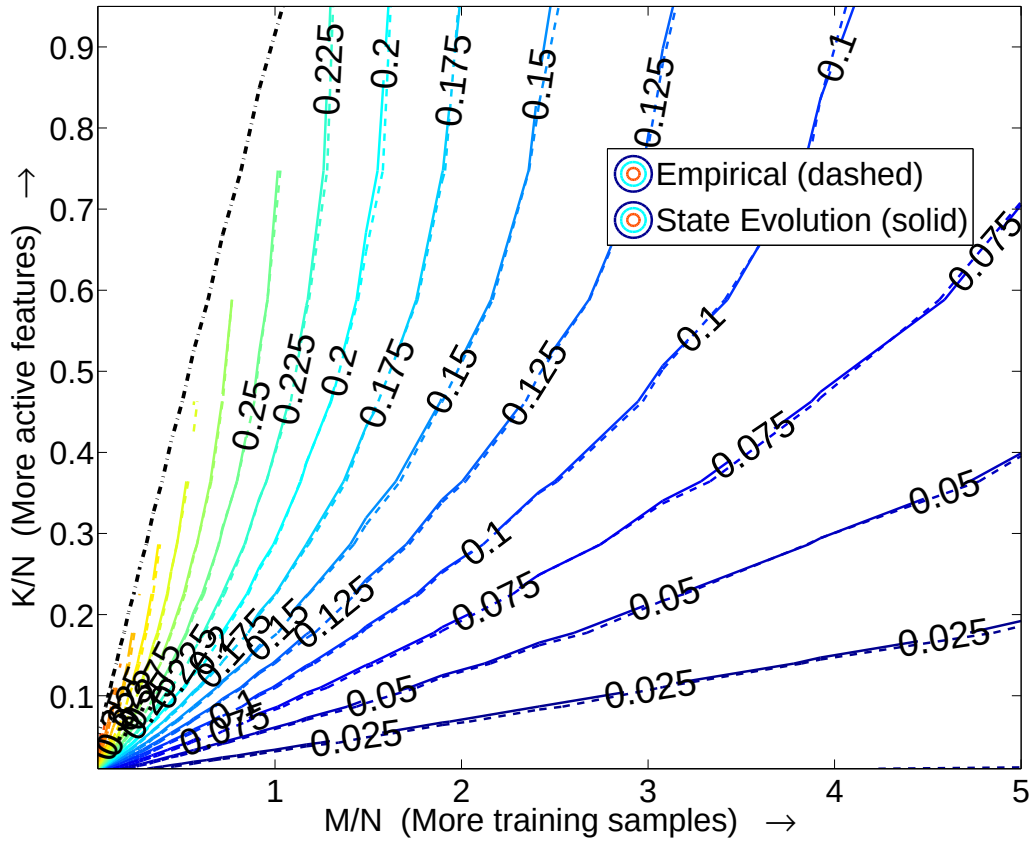
Test-Error Probability via GAMP State Evolution

- Recall that GAMP obeys a state evolution that characterizes the quality of $\hat{\boldsymbol{x}}$ at each iteration t (with i.i.d sub-Gaussian $\boldsymbol{A}$ in the large-system limit).

- We can use this to predict the classification test-error rate.

- For example, with $\boldsymbol{A} \sim$ i.i.d $\mathcal{N}(0,1)$, p_X Bernoulli-Gaussian, $p_{Y|Z}$ probit, we get...

- Notice close agreement between SE (solid) and empirical (dashed).



Robust Classification

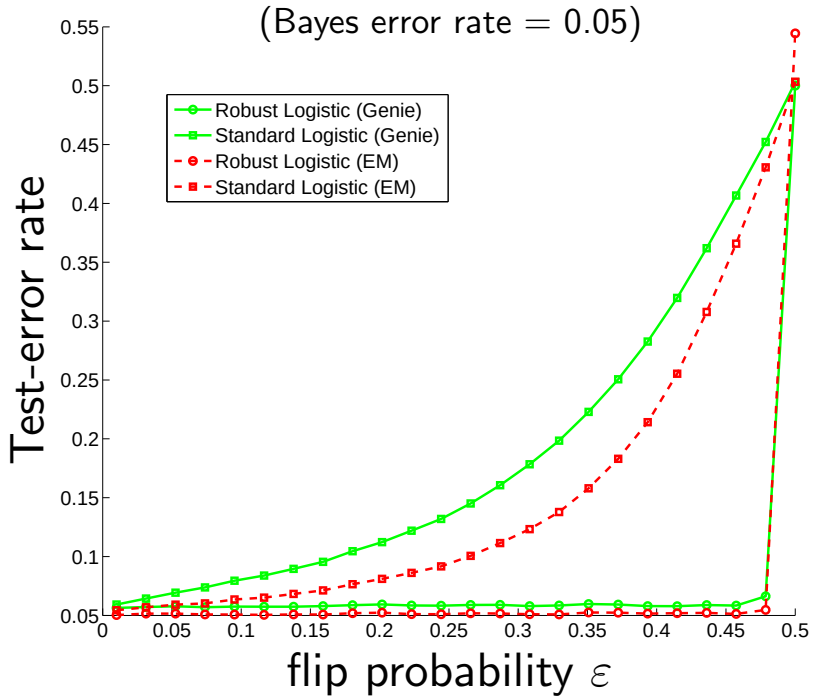
- Some training sets contain corrupted labels (e.g., randomly flipped).
- In response, one can "robustify" any given activation fxn $p_{Y|Z}$ via

$$\tilde{p}_{Y|Z}(y|z) = (1-\varepsilon)p_{Y|Z}(y|z) + \varepsilon\, p_{Y|Z}(1-y|z),$$

  where $\varepsilon \in [0,1]$ models the flip probability.

- The example shows test-error rate for standard (upper) and robust (lower) activation fxns with genie-tuned and EM-tuned ϵ:

- Details: $x_j \sim$ iid $\mathcal{N}(0,1)$, $\boldsymbol{a}_i^{\mathsf{T}}|\{y_i = \pm 1\} \sim$ iid $\mathcal{N}(\pm\boldsymbol{\mu}, \boldsymbol{I})$, ϵ-flipped logistic $p_{Y|Z}$, $m = 8192$, $n = 512$.



Text Classification Example

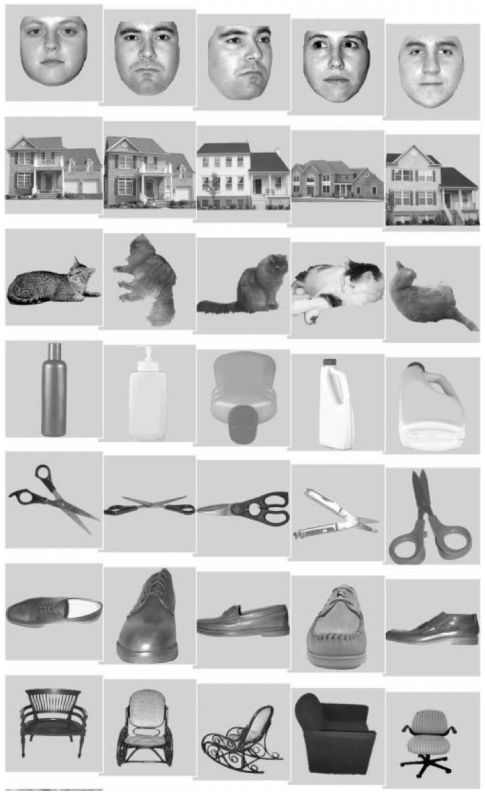
- Reuter's Corpus Volume 1 (RCV1) dataset
- $n = 47\,236$ features, $m = 677\,399$ training examples
- 0.0016-sparse features (far from i.i.d sub-Gaussian $\boldsymbol{A}$!)

Classifier	Tuning	Accuracy	Runtime (s)	Density
spGAMP: BG-PR	EM	97.6%	**317 / 57**	11.1%
spGAMP: BG-HL	EM	**97.7%**	468 / 93	**8.0%**
msGAMP: L1-LR	EM	97.6%	684 / 123	9.8%
CDN	xval	**97.7%**	1298 / 112	10.9%
TRON	xval	**97.7%**	1682 / 133	10.8%
TFOCS: L1-LR	xval	97.6%	1086 / 94	19.2%

$\Rightarrow$ EM-GAMP yields fast, accurate, and sparse classifiers.

Haxby Example

- We now return to the problem of learning, from fMRI measurements, which parts of the brain are responsible for discriminating two classes of object.
- Note that the main problem here is feature selection, not classification. The observed classification error rate is used only to judge the validity of the support estimate.
- For this we use the famous Haxby data, with



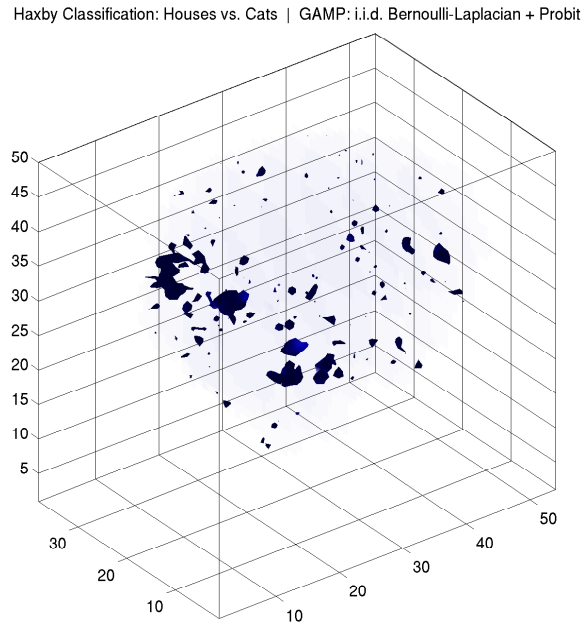
$$n = 31398 \quad \leftrightarrow \quad \text{fMRI voxels}$$
$$m = 216 \quad \leftrightarrow \quad 2 \text{ classes} \times 9 \text{ examples} \times 12 \text{ subjects}$$

Haxby et al., "Distributed and Overlapping Representations of Faces and Objects in Ventral Temporal Cortex" *Science*, 2001.

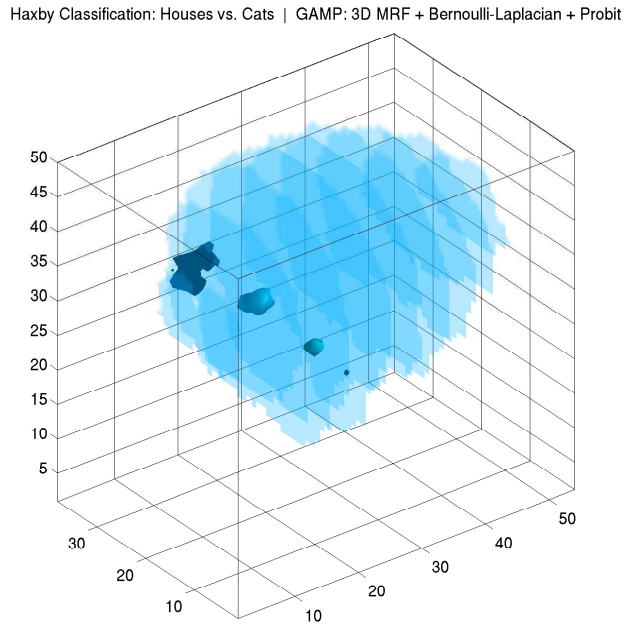
Haxby: Cats vs. Houses

algorithm	setup	error rate	runtime
EM-GAMP	sum-prod logit/B-Laplace	0.9%	9 sec
EM-GAMP	sum-prod probit/B-Laplace	1.9%	13 sec
EM-turbo-GAMP	sum-prod probit/B-Laplace 3D-MRF	2.8%	14 sec

without 3D MRF



with 3D MRF



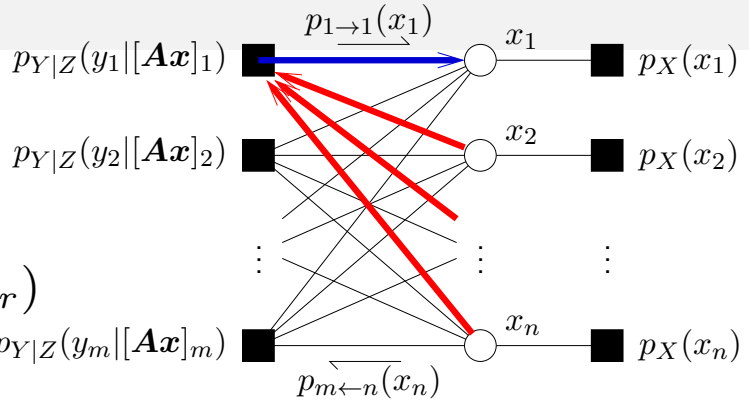
Conclusions

- We presented a novel application of GAMP to binary linear classification & feature selection.
- Some nice properties of classification-GAMP include
  - flexibility in choice of activation function and weight prior
  - runtime (e.g., 3-$4\times$ faster than recent methods)
  - state-evolution can be used to predict test error-rate
  - can handle corrupted labels (via robust prior)
  - can tune without cross-validation (via EM extension)
  - can exploit and learn structured sparsity (via turbo extension)
- All of the above also applies to one-bit compressive sensing.

All these methods are integrated into GAMPmatlab:
`http://sourceforge.net/projects/gampmatlab/`

Thanks!

GAMP Heuristics (Sum-Product)



1. Message from y_i node to x_j node:

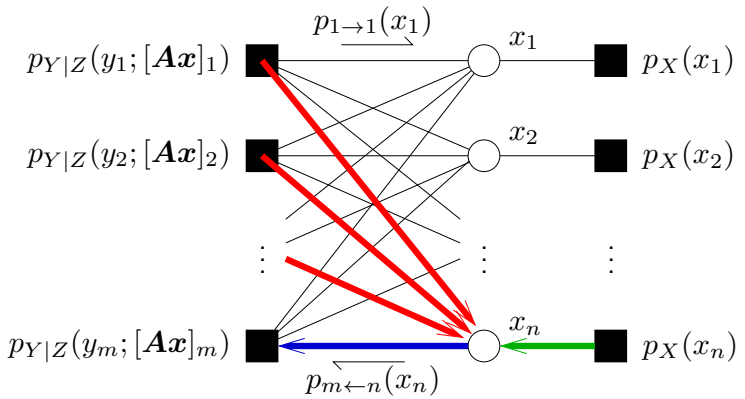
$$p_{i\to j}(x_j) \propto \int_{\{x_r\}_{r\neq j}} p_{Y|Z}\big(y_i; \overbrace{\textstyle\sum_r a_{ir}x_r}^{\approx\mathcal{N}\text{ via CLT}}\big) \prod_{r\neq j} p_{i\leftarrow r}(x_r)$$

$$\approx \int_{z_i} p_{Y|Z}(y_i; z_i)\, \mathcal{N}\big(z_i; \hat{z}_i(x_j), \nu_i^z(x_j)\big) \approx \mathcal{N}$$

To compute $\hat{z}_i(x_j), \nu_i^z(x_j)$, the means and variances of $\{p_{i\leftarrow r}\}_{r\neq j}$ suffice, thus Gaussian message passing!

Remaining problem: we have $2mn$ messages to compute (too many!).

2. Exploiting similarity among the messages $\{p_{i\leftarrow j}\}_{i=1}^m$, GAMP employs a Taylor-series approximation of their difference, whose error vanishes as $m\to\infty$ for dense $\boldsymbol{A}$ (and similar for $\{p_{i\to j}\}_{j=1}^n$ as $n\to\infty$). Finally, need to compute only $\mathcal{O}(m+n)$ messages!



The GAMP Algorithm

Require: Matrix $\boldsymbol{A}$, sum-prod $\in$ {true,false}, initializations $\hat{\boldsymbol{x}}^0$, $\boldsymbol{\nu}_x^0$

$t = 0$, $\hat{\boldsymbol{s}}^{-1} = \boldsymbol{0}$, $\forall ij : S_{ij} = |A_{ij}|^2$

repeat

$\boldsymbol{\nu}_p^t = \boldsymbol{S}\boldsymbol{\nu}_x^t, \quad \hat{\boldsymbol{p}}^t = \boldsymbol{A}\hat{\boldsymbol{x}}^t - \hat{\boldsymbol{s}}^{t-1}.\boldsymbol{\nu}_p^t$ (gradient step)

if sum-prod **then**

$\forall i : \nu_{z_i}^t = \mathsf{var}(Z|P; \hat{p}_i^t, \nu_{p_i}^t), \quad \hat{z}_i^t = \mathsf{E}(Z|P; \hat{p}_i^t, \nu_{p_i}^t),$

else

$\forall i : \nu_{z_i}^t = \nu_{p_i}^t \mathsf{prox}'_{-\nu_{p_i}^t \log p_{Y|Z}(y_i,.)}(\hat{p}_i^t) \quad \hat{z}_i^t = \mathsf{prox}_{-\nu_{p_i}^t \log p_{Y|Z}(y_i,.)}(\hat{p}_i^t),$

end if

$\boldsymbol{\nu}_s^t = (1 - \boldsymbol{\nu}_z^t./\boldsymbol{\nu}_p^t)./\boldsymbol{\nu}_p^t, \quad \hat{\boldsymbol{s}}^t = (\hat{\boldsymbol{z}}^t - \hat{\boldsymbol{p}}^t)./\boldsymbol{\nu}_p^t$ (dual update)

$\boldsymbol{\nu}_r^t = 1./(\boldsymbol{S}^T\boldsymbol{\nu}_s^t), \quad \hat{\boldsymbol{r}}^t = \hat{\boldsymbol{x}}^t + \boldsymbol{\nu}_r^t.\boldsymbol{A}^T\hat{\boldsymbol{s}}^t$ (gradient step)

if sum-prod **then**

$\forall j : \nu_{x_j}^{t+1} = \mathsf{var}(X|R; \hat{r}_j^t, \nu_{r_j}^t), \quad \hat{x}_j^{t+1} = \mathsf{E}(X|R; \hat{r}_j^t, \nu_{r_j}^t),$

else

$\forall j : \nu_{x_j}^{t+1} = \nu_{r_j}^t \mathsf{prox}'_{-\nu_{r_j}^t \log p_X(.)}(\hat{r}_j^t) \quad \hat{x}_j^{t+1} = \mathsf{prox}_{-\nu_{r_j}^t \log p_X(.)}(\hat{r}_j^t),$

end if

$t \leftarrow t+1$

until Terminated

Note connections to Arrow-Hurwicz, primal-dual, ADMM, proximal FB splitting,. . .