Model-based Speech Separation and Recognition

Steven J. Rennie

Chime 2013 Workshop
June 1, 2013
Motivation

- Noise-robust Automatic Speech Recognition (ASR)
- Noise-robust *Multi-talker* ASR
- Signal Separation/Isolation/Analysis/Decomposition

Some Applications

- mobile computing
- surveillance
- acoustic forensics
- signal re-composition/editing
- artificial perception
- enhanced hearing
- robust audio search
Why is Robust ASR hard?

- **Multiple sources of interference, including speech**
  - Computational explosion in the number of possible “acoustic states” of the environment
  - This makes data acquisition difficult
  - This makes statistical data analysis difficult

Combinatorial Considerations

Source Models:
- features $x^n$
- states $s^n$
- number of states $j s^n j$

- functions of connected variables
Combinatorial Considerations

\[ p(s^1) \quad p(s^n) \quad p(s^N) \]

\[ s^1 \quad s^n \quad s^N \]

\[ x^1 \quad x^n \quad x^N \]

Interaction model

\[ p(yjx^1; \phi\phi; x^N) \]

Inference: \[ O(js^n) \]

- functions of connected variables
Factorial Models of Noisy Speech

Signal Models  
Interaction Models

Inference

Predictions

Traffic Noise
Engine Noise
Speech Babble
Airport Noise
Car Noise
Music
Speech
Speech
Exact Interaction: signal with additive noise, log domain

\[ y = \log(e^x + e^n) + \log \left( 1 + \frac{2e^x e^n}{e^x + e^n} \cos(\theta) \right) \]

\[ E_\theta[y|x; n] = \max(x; n) \]


Pascal Speech Separation Challenge:

2006: Factorial HMMs achieve super-human performance on the SSC.

SSC Model

LM to acoustic state model:

\[ p(s_t^a | v_t^a) \]
Beyond Two Sources

2009: New variational methods can separate data with trillions of states

- Excellent separation using a variational posterior with 1K masks/frame

PLACE WHITE AT D ZERO SOON 0 dB

PLACE RED IN H 3 NOW -7 dB

LAY BLUE AT P ZERO NOW -7 dB

PLACE GREEN WITH B 8 SOON -7 dB

Audio demos: http://researcher.watson.ibm.com/researcher/view_project.php?id=2819

Max Interaction: More than two speakers

Given: \( p_{x_f}^k(y_f j s^k); \ \mathbb{C}_{x_f}^k(y_f j s^k) \ \cdot \ p(x_f^k \cdot y_f) \) \( \\forall k \)

Then: \( p(y_f j f s^k g) = \frac{\pm Y}{\pm y_f} \ \mathbb{C}_{x_f}^k(y_f j s^k) \)

\[ = \begin{cases} p_{x_f}^k(y_f j s^k) \ \mathbb{C}_{x_f}^j(y_f j s^j) \\ \mathbb{X} \end{cases} \]

\[ = p(y_f ; d_f j f s^k g) \]

\[ \log(p(y_f j f s^k g)) = \phi(d_f) \log \frac{p(y_f ; d_f j f s^k g)}{q(d_f)} \]

*If all masks are known, the sources can be independently inferred.
Max Interaction: New Variational Bound

\[
\log(p(y_f \mid f s^k g)) = \log \left( \prod_{d_f} p(y_f ; d_f \mid f s^k g) \right) \\
= \sum_{d_f} \sum_{s^k} \log \left( \frac{p(y_f ; d_f \mid f s^k g)}{q(d_f)} \right) q(d_f)
\]

- If \( q(d_f) = p(d_f \mid y_f ; f s^k g) \) the bound is tight!
- Complexity of inference (i.e. #masks inferred) can be controlled
- Models of the sources are utilized to \textit{jointly estimate the masks and decode the sources}
- \textit{Deep connections with CASA and MFT.}

Two Speaker Results

![Graph showing word error rates for different methods](image)
Factorial RBMs for robust ASR

**Motivation:**

- Learn parts-based models
  - Distributed states
    - Compositional model
    - Better generalization

- Leverage known interactions
  - Instead of learning the transformation from noisy speech to clean speech again and again
Review: Restricted Boltzmann Machines

- **A Markov Random Field (MRF)**
  - Two layers, no connections between hidden layer nodes
  - For binary hidden, Gaussian visible units:

\[
\log p(v; h) = \sum_{i=1}^{X^v} \left( \frac{(v_i - h_i)^2}{2^{3/2}} \right) + \sum_{j=1}^{X^h} a_j h_j + \sum_{i=1}^{X^v} \sum_{j=1}^{X^h} v_i h_j \frac{a_j + \sum_{i=1}^{X^v} v_i}{Z}
\]

- Form of conditional posterior of hidden units

\[
p(h_j = 1 | v) = \frac{\exp(a_j + \sum_{i=1}^{X^v} v_i h_j)}{1 + \exp(a_j + \sum_{i=1}^{X^v} v_i h_j)}
\]

\[
= \text{sig}(a_j + \sum_{i=1}^{X^v} v_i h_j)
\]
Review: Restricted Boltzmann Machines (cont’d)

- Form of conditional prior of a visible unit

\[
p(v_i | h) = \frac{ \exp( \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{H} \sum_{ij} v_i h_j p_h )}{ \sum_i \exp( \sum_j \frac{(v_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{H} \sum_{ij} v_i h_j p_h )}
\]

\[
= N(v_i; b_i + \sum_{j=1}^{H} \sum_{ij} h_j \sigma_i^2; \sigma_i^2)
\]

- Can be represented as a mixture of \(2^H\) Gaussians

- Can be evaluated in time \textit{linear} in the number of hidden units \(H\) since

\[
p(h_j | v) = \sum_i p(h_j | v_i) = \sum_i p(h_j | v_i)
\]
Factorial Hidden Restricted Boltzmann Machines

- *Interaction Model* $p(y_j \mid v^x; v^n)$ describes how the visible units of multiple RBMs (two here) generate observed data.
- Inference now *intractable* due to explaining away effects.
- One solution: *variational methods*

\[
\log p(y) = \log \prod_{h;v} p(h^x; v^x)p(h^n; v^n)p(y_j v^x; v^n)
\]

\[
= \sum_{h;v} \log \frac{p(h^x; v^x)p(h^n; v^n)p(y_j v)}{q(h; v)} \cdot \log \frac{p(h^x; v^x)p(h^n; v^n)p(y_j v)}{q(h; v)}
\]

\[
= \mathbb{E}_{q(v^x; v^n)}[\log p(y_j v)] + \sum_{i=x; n} \mathbb{E}_{q(h^i; v^i)}[\log \frac{p(h^i; v^i)}{q(h^i; v^i)}] \cdot L
\]

- Choose *surrogate posterior* $q$ that makes inference tractable.
  (bound tight without structural assumptions on $q$)
FHRBM Model: Factor Graph

**Speech Model**

- $l^x$
- $h^x$
- $v^x$

**Noise Model**

- $l^n$
- $h^n$
- $v^n$

**Interaction Model**

$$p(y_j v^x; v^n)$$

Noisy Data $\rightarrow y$
FHRBMs for Robust ASR

- Speech RBM \( p(v^x; h^x) \)
- Noise RBM \( p(v^n; h^n) \)
- Interaction Model (log Mel power spectrum)
  \[
p(y_j v^x; v^n) = \mathcal{N} (y_f ; g(v_f) ; \tilde{A}_f^2) ; \quad v_f = [v_f^x, v_f^n]^T
\]
  \[
g(v_f) = \log(\exp(v_f^x) + \exp(v_f^n)) \quad \text{[this choice ignores phase interactions]}
\]
- Assumed form of surrogate posterior \( q \)
  \[
  q(h^x; v^x; h^n; v^n) = \prod_{j=1}^Y q(v_f^x; v_f^n) q(h_j^x) q(h_k^n)
  \]
  \[
  = \prod_{s=x, n}^{\mathcal{Y}} \mathcal{N} (v_f ; \hat{v}_f ; \hat{G}_f) (o_{h_j^s})_{h_j^s} (1_i o_{h_j^s})_{h_j^s}^{1_i}
  \]
FHRBMs for Robust ASR

- Iteration:
  1. Update context-dependent linear approx. of interaction

\[
p(y_j v^x; v^n) \frac{1}{4} N(y_f; g^{1_f}) + (v_f \mid 1_f)^T d_f; \tilde{A}_f^2);
\]

\[
d_f = [d_{v_f}^x \ d_{v_f}^n]^T = \frac{\partial g}{\partial v_f} v_f = 1_f
\]

\[
d_{v_f}^x = \text{sig}(1_{v_f} \mid 1_{v_f}^x)
\]

\[
d_{v_f}^n = 1 \mid d_{v_f}^x
\]
Model-based Speech Separation and Recognition

Joint Prior, $p(x,n)$

Likelihood, $p(y|x,n)$

Posterior, $p(x,n|y)$

Joint Prior, $p(x,n)$

Approximate Likelihood, $q(y|x,n)$

Approximate Posterior, $q(x,n|y)$
Model-based Speech Separation and Recognition
FHRBMs for Robust ASR

Iteration:

2. Update the variational parameters of source \( s \)

\[
\hat{A}_{vf}^s = (\frac{3}{4} A_{vf}^s + d_{vf}^s (\tilde{A}_f^0)^2) i^{1}
\]

\[
\tilde{v}_{vf}^s = \hat{A}_{vf}^s (\frac{3}{4} b_{vf}^s + \frac{3}{2} \sum_{j=1}^{P} H_{vf}^s \circ h_{vf}^s) + d_{vf}^s (\tilde{A}_f^0)^2 y_f^0
\]

\[
y_f^0 = y_f \circ g_{vf} \circ v_f^s
\]

\[
\tilde{A}_f^0 = \tilde{A}_f + g_{vf} \circ v_f^s
\]

\[
\circ h_{vf}^s = \text{sign}(a_{vf}^s + \sum_{f=1}^{P} s_{vf}^s v_f^s)
\]

3. Toggle \( s \) (between \( s=x \) and \( s=n \))

Deep FHRBM for Robust ASR

- Updates readily generalize to use of deep belief network (DBNs) of RBMs
- Example: Source RBMs with two hidden layers
  - Top Layer Variables
    \[ l^S = f(l^S_1; l^S_k; \ldots; l^S_L) \]
  - Variational distribution
    \[ q(l^S) = \prod_k q(l^S_k) = \prod_k o_l^S \]
  - New update for first hidden layer
    \[ o_{h^S_j} = \text{sig}(a^S_{j_1} + \sum_{i=1}^L v^S_{i_1} l^S_i + \sum_{k=1}^S o_{l^S_k}) \]

Influence of layer below
Influence of layer above

- Extension to use of source RBMs with more than two hidden layers straightforward…
Model-based Speech Separation and Recognition

Speech RBM

Middle Layer (32 Units)

Top Layer (8 Units)

Features (24 dim)

Noise RBM

Middle Layer (8 Units)

Top Layer (3 Units)

Features (24 dim)
Experimental Results

- **Task:** Test time only noise compensation, noisy in-car speech data
- **Recognizer:** IBM embedded system (eVV)
- **AM:** 10K Gaussians, 865 CD states
- **LM:** task-specific grammars
- **Training data:** 786 hrs, ~10K speakers, C&C, dialing, navigation
- **Test data:** 206k words, *well matched*
Results (cont’d.)

- **WER/SER Ranks**
  - 1
  - 2
  - 3

- **jθ_{RB_M} x j = jθ_{GM_M} M x j**

- DNA outperforms use of noise GMM on this task (diffuse evolving noise)

- FHRBM outperforms DNA, but not DNA with Condition Detection (CD)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speech Model</th>
<th>Noise Model</th>
<th>WER/SER (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fMMI (B1)</td>
<td>GMM</td>
<td>Gaussian Process</td>
<td>1.34/3.77</td>
</tr>
<tr>
<td>B1 + SNM</td>
<td></td>
<td></td>
<td>1.70/5.06</td>
</tr>
<tr>
<td>B1 + DNA</td>
<td></td>
<td>3</td>
<td>1.27/4.04</td>
</tr>
<tr>
<td>B1 + FHRBM</td>
<td></td>
<td>2</td>
<td>1.20/3.51</td>
</tr>
<tr>
<td>B1 + DNA-CD</td>
<td></td>
<td>1</td>
<td>1.09/3.19</td>
</tr>
<tr>
<td>fMMI+SS (B2)</td>
<td></td>
<td>2</td>
<td>1.18/3.41</td>
</tr>
<tr>
<td>B2 + SNM</td>
<td></td>
<td></td>
<td>1.76/5.27</td>
</tr>
<tr>
<td>B2 + DNA</td>
<td></td>
<td>3</td>
<td>1.34/4.24</td>
</tr>
<tr>
<td>B2 + FHRBM</td>
<td>2</td>
<td>1.18/3.48</td>
<td></td>
</tr>
<tr>
<td>B2 + DNA-CD</td>
<td>1</td>
<td>1.10/3.17</td>
<td></td>
</tr>
<tr>
<td>fMMI+fMLLR (B3)</td>
<td></td>
<td></td>
<td>1.08/3.00</td>
</tr>
<tr>
<td>B3 + SNM</td>
<td></td>
<td>3</td>
<td>1.25/3.59</td>
</tr>
<tr>
<td>B3 + DNA</td>
<td>2</td>
<td>1.06/3.04</td>
<td></td>
</tr>
<tr>
<td>B3 + FHRBM</td>
<td>2</td>
<td>1.03/2.95</td>
<td></td>
</tr>
<tr>
<td>B3 + DNA-CD</td>
<td>1</td>
<td>0.93/2.59</td>
<td></td>
</tr>
<tr>
<td>fMMI+fMLLR+SS (B4)</td>
<td>2</td>
<td></td>
<td>1.00/2.79</td>
</tr>
<tr>
<td>B4 + SNM</td>
<td></td>
<td></td>
<td>1.26/3.56</td>
</tr>
<tr>
<td>B4 + DNA</td>
<td>3</td>
<td>1.02/3.03</td>
<td></td>
</tr>
<tr>
<td>B4 + FHRBM</td>
<td>3</td>
<td>0.99/2.82</td>
<td></td>
</tr>
<tr>
<td>B4 + DNA-CD</td>
<td>1</td>
<td>0.95/2.67</td>
<td></td>
</tr>
</tbody>
</table>
Results (cont’d.) – WER vs. (biased) SNR

Per-SNR word errors. fMMI model, fMLLR off, SS off
Results (cont’d.) – WER vs. (biased) SNR

Per-SNR word errors. fMMI model, fMLLR on, SS on
Thoughts

- **Results/investigations quite preliminary**
  - **Models**
    - DNA: (matched) quasi-stationary noise model, speech GMM
    - FHRBM: no dynamics yet, tiny RBMs
  - **SNR estimates of each frequency band**
    - DNA: estimated uniquely for every speech state for each frame
    - FHRBM: single set of SNR estimates for each frame
  - **Initialization**
    - DNA: noise model initialized on first 10 frames
    - FHRBM: only state posterior (feature layer not yet adapted)
  - **Need to evaluate FHRBMs on more general noise containing non-stationary & structured elements**
  - **Need to explore model/inference procedures further: e.g. FHRBM a bootstrap for fast feed-forward system?**
Direct Product Based Deep Neural Networks

Motivation:

- **Resurgence of interest/success with DNNs for ML**
  - New algorithms, more data, better machines
- **Still time-consuming to train**
  - Restricts neurons/layer, #layers utilized

Idea:

- **Learn networks with connections that can be represented using sums of direct products**
  - Make it feasible to learn networks with **millions** of neurons
Direct Product DNNs

- Constrain the weight matrix $W$ to be a sum of \textit{direct products}

\[ W = \bigotimes_{i} A_{i} \odot B_{i}; \]

- Direct products: Kronecker, outer, “box” product
- Low rank $W$ a DPDNN, Input layers are naturally Kronecker-structured for spliced input data
- A structured weight-tying strategy that
  - Facilitates efficient matrix multiplication, storage
  - Composes $W$ from sets of “complete” bases: exact representation always possible
Review

- **The Kronecker Product (KP)**

\[
W = A \otimes B = \begin{pmatrix}
a_{11}B & a_{12}B \\
a_{21}B & a_{22}B
\end{pmatrix}
\]

**Interesting Facts:**

- FFT can be expressed as a recursive factorization of the DFT matrix using KPs
- So can several combinatorial algorithms
- Any circulant Matrix can be diagonalized by DFT
Kronecker Product DNNs

\[ W = \bigotimes_{i} A_i \otimes B_i; \]

- **Efficient Matrix Multiplication**

\[(A_i \otimes B_i)\text{vec}(Z) = \text{vec}(B_iZA_i^T)\]

- For \(M=N\), A, B square \(O(N^2)\) ! \(O(N^3=2)\)

- E.g. Multiplying a vector by a 10K x 10K matrix requires only 1 million rather than 100 million scalar multiplications.

- **Efficient Storage**

- For \(M=N\), all A,B square \(O(N^2)\) ! \(O(2N)\)

- A 1M x 1M matrix has only 2 million parameters, rather than 1 trillion
Factoring Existing DBNs

- Recall

\[ W = \bigotimes_{i} A_{i} \otimes B_{i}; \quad A_{i} \in \mathbb{R}^{M_{i} \times N_{i}}; \quad B_{i} \in \mathbb{R}^{O_{i} \times P_{i}} \]

- If all A, B dims are independent of i, reduces to an SVD problem (Van Loan, 1992)

\[ W = \bigotimes_{i} A_{i} \otimes B_{i} = U D V^{T} \]

- Spliced features lead to Kron-structured W

Learning/Inference

- **Forward Pass**

\[
    z_j = \frac{3}{4} x_j = \frac{3}{4} W z_{j-1} + b_j
\]

\[
    x_j = \text{vec}(B_i Z_{j-1}^T A_i^T) + b_j
\]

- **Error Back-propagation**

\[
    \delta_{i-1} = \frac{3}{4} \delta_j X
    = \frac{3}{4} (x_{j-1}) \odot W^T \delta_j
    \]

\[
    \frac{\partial E}{\partial A_i} = \delta_{(j)^T} B_i Z_{j-1}^T;
    \frac{\partial E}{\partial B_i} = \delta_{j} A_i Z_{j-1}^T
\]
Experiments

- 50 hr English Broadcast News (EBN) task
- Training: 50 hours 1996/1997 EBN (5/50 Dev.)
- Test: 3 hrs EARS dev-04f set

Acoustic Model
- Hybrid (NN fully replaces GMM)
- 2200 acoustic targets
- Features: 13 dim. PLP -> VTLN -> CMS -> splice ±4 frames -> 117 dim. input features

Baseline:
- NN topology: 117 -> 1K -> 1K -> 2200
- NN training: Stochastic Gradient, CE (no pre-training)
- WER: 23.0%
Training DPDBNs

- **Poor Man’s Trainer:**
  - Enforce Kronecker structure via periodic SVD during training (first layer only)
  - Not effective

\[
W = \bigotimes_{i} A_i \otimes B_i;
\]

\[
W = 2^R 1024^L 117
\]

\[
A_i = 2^R 1^L 9
\]

\[
B_i = 2^R 1024^L 13
\]

<table>
<thead>
<tr>
<th>terms</th>
<th>%FAcc</th>
<th>%WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34.1</td>
<td>24.3</td>
</tr>
<tr>
<td>2</td>
<td>33.7</td>
<td>25.0</td>
</tr>
<tr>
<td>3</td>
<td>34.0</td>
<td>24.5</td>
</tr>
<tr>
<td>all (baseline)</td>
<td>35.0</td>
<td>23.0</td>
</tr>
</tbody>
</table>
Training DPDBNs

- **By Projected Gradient**
  - Project full gradient onto representation
  - Pros: easy, correct, sub-routine of existing trainer
  - Cons: no training speedup, can’t train large \( W \)

<table>
<thead>
<tr>
<th>L1 Topology</th>
<th>FAcc</th>
<th>WER</th>
<th>L1 Param. Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1024,117] (base)</td>
<td>35.0</td>
<td>23.0</td>
<td>1</td>
</tr>
<tr>
<td>1x[32<em>9, 32</em>13]</td>
<td>32.8</td>
<td>25.0</td>
<td>170</td>
</tr>
<tr>
<td>2x[32<em>9, 32</em>13]</td>
<td>32.9</td>
<td>24.9</td>
<td>85</td>
</tr>
<tr>
<td>3x[32<em>9, 32</em>13]</td>
<td>33.2</td>
<td>24.6</td>
<td>57</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Topology L1/L2/L3</th>
<th>PR L1/L2/L3</th>
<th>PR DNN</th>
<th>FAcc</th>
<th>WER</th>
<th>Topology L1/L2/L3</th>
<th>PR L1/L2/L3</th>
<th>PR DNN</th>
<th>FAcc</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>(740,117)</td>
<td>1.4</td>
<td>1.5</td>
<td>32.7</td>
<td>26.4</td>
<td>5x(32<em>32, 9</em>13)</td>
<td>27</td>
<td>1.5</td>
<td>33.7</td>
<td>24.8</td>
</tr>
<tr>
<td>(740,740)</td>
<td>1.9</td>
<td></td>
<td></td>
<td></td>
<td>10x(32<em>32, 32</em>32)</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2220,740)</td>
<td>1.4</td>
<td></td>
<td></td>
<td></td>
<td>(2220,1024)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(280,117)</td>
<td>3.7</td>
<td>4.7</td>
<td>31.2</td>
<td>27.7</td>
<td>5x(32<em>32, 9</em>13)</td>
<td>27</td>
<td>4.7</td>
<td>31.9</td>
<td>26.9</td>
</tr>
<tr>
<td>(280,280)</td>
<td>13.4</td>
<td></td>
<td></td>
<td></td>
<td>10x(32<em>32, 32</em>32)</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2220, 280)</td>
<td>3.7</td>
<td></td>
<td></td>
<td></td>
<td>10x(2220<em>1, 32</em>32)</td>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(135,117)</td>
<td>7.6</td>
<td>10.3</td>
<td>28.8</td>
<td>31.2</td>
<td>5x(32<em>32, 9</em>13)</td>
<td>27</td>
<td>10.3</td>
<td>29.8</td>
<td>29.1</td>
</tr>
<tr>
<td>(135,135)</td>
<td>57.3</td>
<td></td>
<td></td>
<td></td>
<td>20x(32<em>32, 32</em>32)</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2220, 135)</td>
<td>7.6</td>
<td></td>
<td></td>
<td></td>
<td>4x(2220<em>1, 32</em>32)</td>
<td>7.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2k, 117)</td>
<td>0.5</td>
<td>0.75</td>
<td>34.7</td>
<td>23.5</td>
<td>5x(64<em>64, 9</em>13)</td>
<td>11</td>
<td>1.5</td>
<td>35.0</td>
<td>23.5</td>
</tr>
<tr>
<td>(1k, 2k)</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td>10x(64<em>64, 64</em>64)</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2220, 1k)</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td>(2220,1024)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Trend:** For fixed params, DPDNNs outperform standard DNNs.
Current/Future Work

- **Native DPDBN Trainer**
  - Operational, experimentation in progress
    - Currently training 100K by 100K weight matrices

- **Generalization of estimation framework**
  - to non-uniform direct products of non-uniform size, and transformations thereof

- **Test interactions**
  - RELU, dropout, input noisification,…

- **Investigate on composite data**
  - Factorizations correspond to independence assumptions…
Scalar-Matrix Function Optimization

- The Anatomy of the Hessian

**Theorem:** Any scalar-matrix function $f(X)$ formed using trace($\cdot$), log det($\cdot$), $(\cdot)^T$, and arithmetic operations ($+$, $-$, $\ast$ and $(\cdot)^{-1}$) has a Hessian of the form:

$$f''(X) = \sum_{i=1}^{k_1} A_i \otimes B_i + \sum_{i=k_1+1}^{k_2} A_i \boxtimes B_i + \sum_{i=k_2+1}^{k} \text{vec}(A_i)\text{vec}^T(B_i),$$

This allows the Hessian to be efficiently utilized...

$$f''(X)\text{vec}(C) = \text{vec}\left(\sum_{i=1}^{k_1} B_i C A_i^T + \sum_{i=k_1+1}^{k_2} B_i C^T A_i^T + \sum_{i=k_2+1}^{k} A_i \text{trace}(C^T B_i)\right).$$

Scalar Matrix Function Optimization

- **Efficiently Solving the Covariance Selection Problem**
  - Problem: infer a sparse (L1-regularized) inverse covariance matrix that maximizes the probability of a dataset
    \[
    P^* = \arg \max_{P \succeq 0} \log \det(P) - \text{trace}(SP) - \lambda \| \vec(P) \|_1,
    \]
  - Approach: Iteratively apply Newton-like algorithms on locally quadratic approximations to the objective
    - Efficient inference by exploiting sparsity & structure of Hessian
  - Applicability: recently shown that covariance selection can be used to infer the structure of more general networks (e.g. discrete)

Closing Remarks

- **Questions to ponder**
  - The evolving role of models that can “explain away” phenomena
    - Are feed-forward representations sufficient?
  - The known and still poorly understood limitations of current neural networks
    - More teaching, less tuning
  - The increasingly important role of optimization methods in machine learning and signal processing
    - Help the machine help itself
  - The role of separation and robustness research in ASR
    - Commercial systems are now *very* good, but the NN revolution is blurring the distinction between core and robust ASR.
Thank-you.