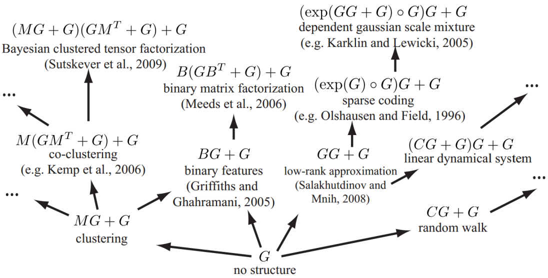
# Automated Model Construction through Compositional Grammars





David Duvenaud

Cambridge University
Computational and Biological Learning Lab


April 22, 2013

# OUTLINE

- Motivation
- Automated structure discovery in regression
  - Gaussian process regression
  - Structures expressible through kernel composition
  - A massive missing piece
  - grammar & search over models
  - Examples of structures discovered
- Automated structure discovery in matrix models
  - expressing models as matrix decompositions
  - grammar & special cases
  - examples of structures discovered on images

# CREDIT WHERE CREDIT IS DUE

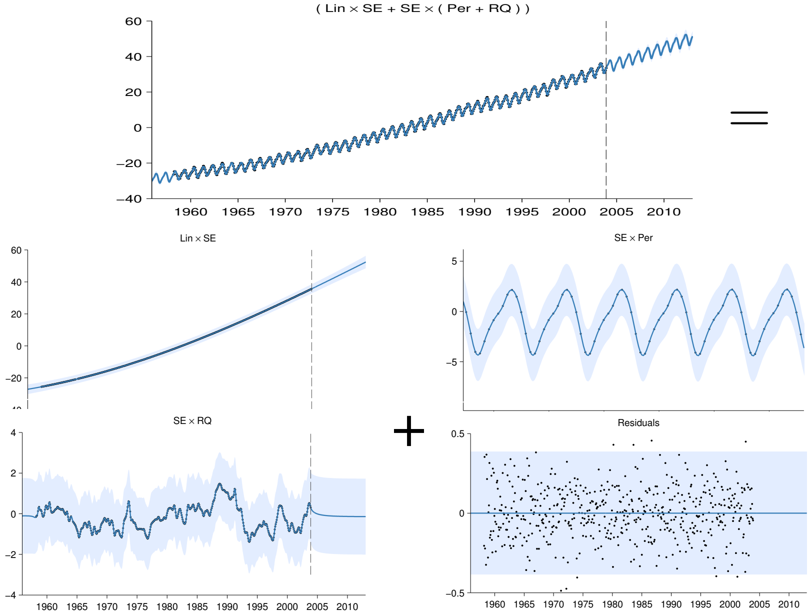
Talk based on two papers:

- Structure Discovery in Nonparametric Regression through Compositional Kernel Search [ICML 2013]
  David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, Zoubin Ghahramani
- Exploiting compositionality to explore a large space of model structures [UAI 2012]
  Roger B. Grosse, Ruslan Salakhutdinov, William T. Freeman, Joshua B. Tenenbaum

# MOTIVATION

- Models today built by hand, or chosen from a fixed set.
  - Fixed set sometimes not that rich
    - Just being nonparametric sometimes isn't good enough
    - to learn efficiently, need to have a rich prior that can express most of the structure in your data.
  - Building by hand requires expertise, understanding of the dataset.
  - Follows cycle of: propose model, do inference, check model fit
    - Propose new model
    - Do inference
    - Check model fit
- Andrew Gelman asks: How would an AI do statistics?
- It would need a language for describing arbitrarily complicated models, a way to search over those models, nad a way of checking model fit.

# Finding Structure in GP Regression



# Gaussian Process Regression

Assume $\mathbf{X}, \mathbf{y}$ is generated by $\mathbf{y} = \mathbf{f}(\mathbf{X}) + \epsilon_\sigma$
A GP prior over $\mathbf{f}$ means that, for any finite set of points $\mathbf{X}$,

$$p(\mathbf{f}(\mathbf{x})) = \mathcal{N}(\mu(\mathbf{X}), K(\mathbf{X}, \mathbf{X}))$$

where

$$K_{ij} = k(\mathbf{X}_i, \mathbf{X}_j)$$

is the *covariance function* or *kernel*. $k(x, x') = \operatorname{cov}[f(x), f(x')]$

# Gaussian Process Regression

Assume $\mathbf{X}, \mathbf{y}$ is generated by $\mathbf{y} = \mathbf{f}(\mathbf{X}) + \epsilon_\sigma$
A GP prior over $\mathbf{f}$ means that, for any finite set of points $\mathbf{X}$,

$$p(\mathbf{f}(\mathbf{x})) = \mathcal{N}(\mu(\mathbf{X}), K(\mathbf{X}, \mathbf{X}))$$

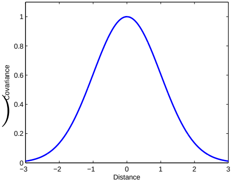
where

$$K_{ij} = k(\mathbf{X}_i, \mathbf{X}_j)$$

is the *covariance function* or *kernel*. $k(x, x') = \text{cov}\left[f(x), f(x')\right]$

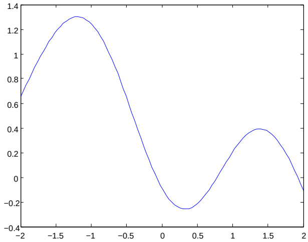
Typically, kernel says that nearby $x_1, x_2$ will have highly correlated function values $f(x_1), f(x_2)$:

$$k_{\text{SE}}(x, x') = \exp(-\frac{1}{2\theta}|x - x'|_2^2)$$



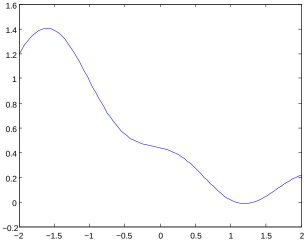
# Sampling from a GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
    % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

    % Specify that the prior mean of f is zero.
    mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



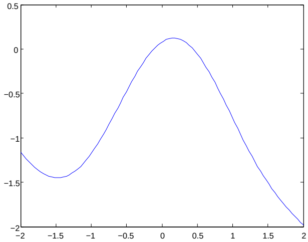
# SAMPLING FROM A GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
    % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

    % Specify that the prior mean of f is zero.
    mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



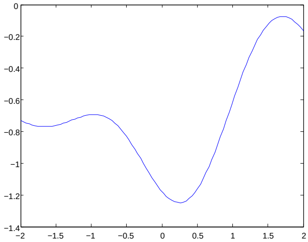
# SAMPLING FROM A GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
    % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

    % Specify that the prior mean of f is zero.
    mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



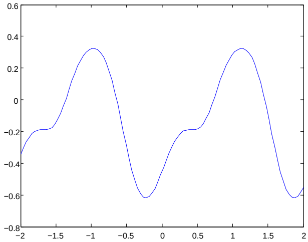
# SAMPLING FROM A GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
    % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

    % Specify that the prior mean of f is zero.
    mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



# Sampling from a GP

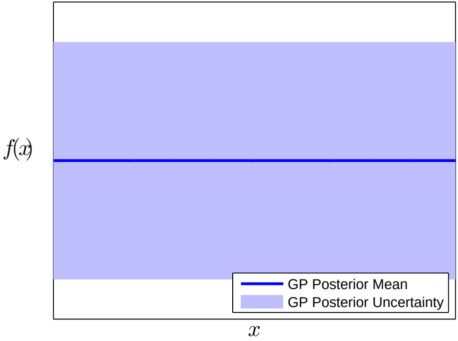
```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
    % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

    % Specify that the prior mean of f is zero.
    mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Periodic covariance function.
function c = covariance(x, y)
    c = exp( -0.5*( sin(( x - y )*1.5).^2 ));
end
```



# Conditional Posterior

$$f(x^*)|\mathbf{X},\mathbf{y} \sim \mathcal{N}(k(x^*,\mathbf{X})K^{-1}\mathbf{y},\\ k(x^*,x^*) - k(x^*,\mathbf{X})K^{-1}k(\mathbf{X},x^*))$$

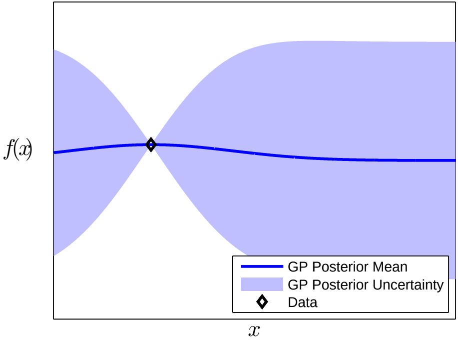
With SE kernel:



# Conditional Posterior

$$f(x^*)|\mathbf{X},\mathbf{y} \sim \mathcal{N}(k(x^*,\mathbf{X})K^{-1}\mathbf{y}, \\ k(x^*,x^*) - k(x^*,\mathbf{X})K^{-1}k(\mathbf{X},x^*))$$

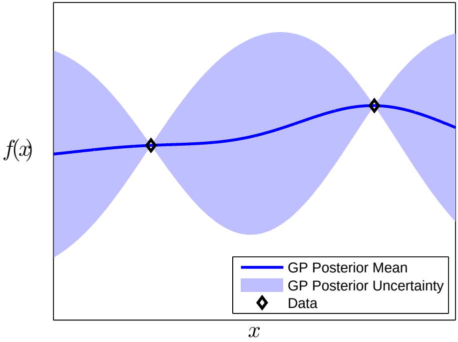
With SE kernel:



# Conditional Posterior

$$f(x^*)|\mathbf{X},\mathbf{y} \sim \mathcal{N}(k(x^*,\mathbf{X})K^{-1}\mathbf{y},$$
$$k(x^*,x^*) - k(x^*,\mathbf{X})K^{-1}k(\mathbf{X},x^*))$$

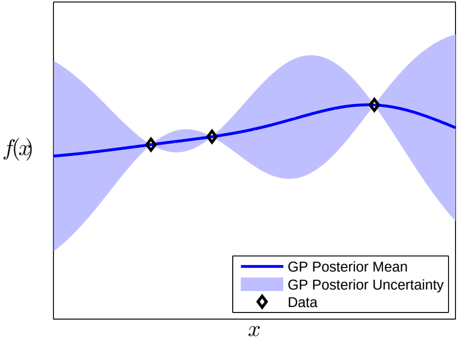
With SE kernel:



# Conditional Posterior

$$f(x^*)|\mathbf{X},\mathbf{y} \sim \mathcal{N}(k(x^*,\mathbf{X})K^{-1}\mathbf{y},$$
$$k(x^*,x^*) - k(x^*,\mathbf{X})K^{-1}k(\mathbf{X},x^*))$$

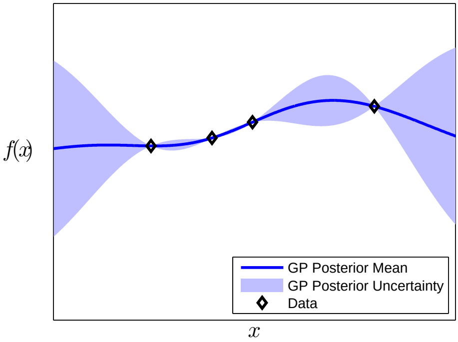
With SE kernel:



# Conditional Posterior

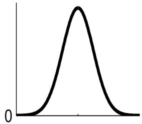
$$f(x^*)|\mathbf{X},\mathbf{y} \sim \mathcal{N}(k(x^*,\mathbf{X})K^{-1}\mathbf{y},\\ k(x^*,x^*) - k(x^*,\mathbf{X})K^{-1}k(\mathbf{X},x^*))$$

With SE kernel:

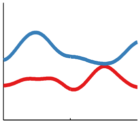


# Kernel Choice is Important

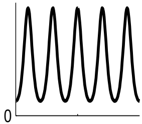
- Kernel determines almost all the properties of the prior.
- Many different kinds, with very different properties:



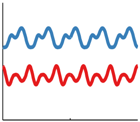

Squared-exp (SE)



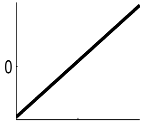
local variation



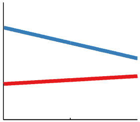

Periodic (PER)



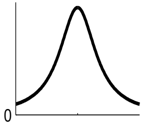
repeating structure



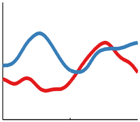

Linear (LIN)



linear functions



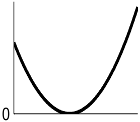

Rational-quadratic(RQ)



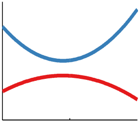
multi-scale variation

# Kernels can be composed

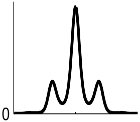
- Two main operations: adding, multiplying



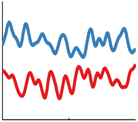

LIN × LIN



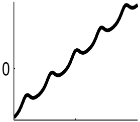
quadratic functions



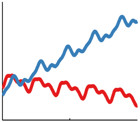

SE × PER



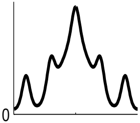
locally periodic



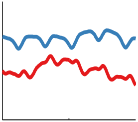

LIN + PER



periodic with trend



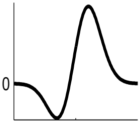

SE + PER



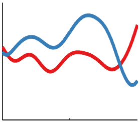
periodic with noise

# Kernels can be composed

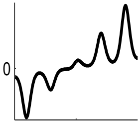
- Can be composed across multiple dimensions



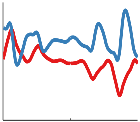

Lin × SE



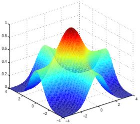
increasing variation



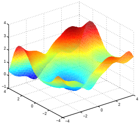

Lin × Per



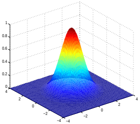
growing amplitude



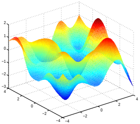
$\text{SE}_1 + \text{SE}_2$



$f_1(x_1) + f_2(x_2)$



$\text{SE}_1 \times \text{SE}_2$



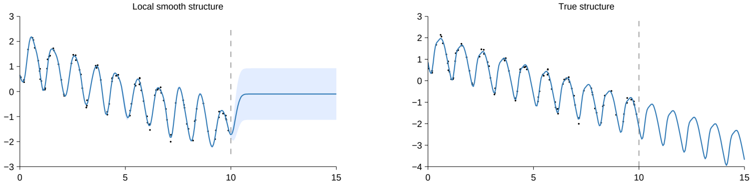
$f(x_1, x_2)$

# SPECIAL CASES

| | |
|---|---|
| Bayesian linear regression | LIN |
| Bayesian polynomial regression | LIN $\times$ LIN $\times \ldots$ |
| Generalized Fourier decomposition | PER $+$ PER $+ \ldots$ |
| Generalized additive models | $\sum_{d=1}^{D} \mathrm{SE}_d$ |
| Automatic relevance determination | $\prod_{d=1}^{D} \mathrm{SE}_d$ |
| Linear trend with deviations | LIN $+$ SE |
| Linearly growing amplitude | LIN $\times$ SE |

# Appropriate kernels are necessary for extrapolation

- SE kernel $\rightarrow$ basic smoothing.
- Richer kernels means richer structure can be captured.



# Kernels are hard to choose

- Given the diversity of priors available, how to choose one?
- Standard GP software packages include many base kernels and means to combine them, but *no default kernel*
- Software can't choose model for you, you're the expert (?)

# Kernels are hard to construct

- Carl devotes 4 pages of his book to constructing a custom kernel for CO2 data
- requires specialized knowledge, trial and error, and a dataset small and low-dimensional enough that a human can interpret it.
- In practice, most users can't or won't make custom kernel, and SE kernel became *de facto* standard kernel through inertia.

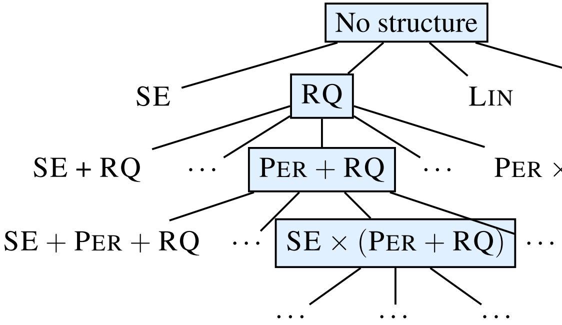
# RECAP

- GP Regression is a powerful tool
- Kernel choice allows for rich structure to be captured - different kernels express very different model classes
- Composition generates a rich space of models
- Hard & slow to search by hand
- Can kernel specification be automated?

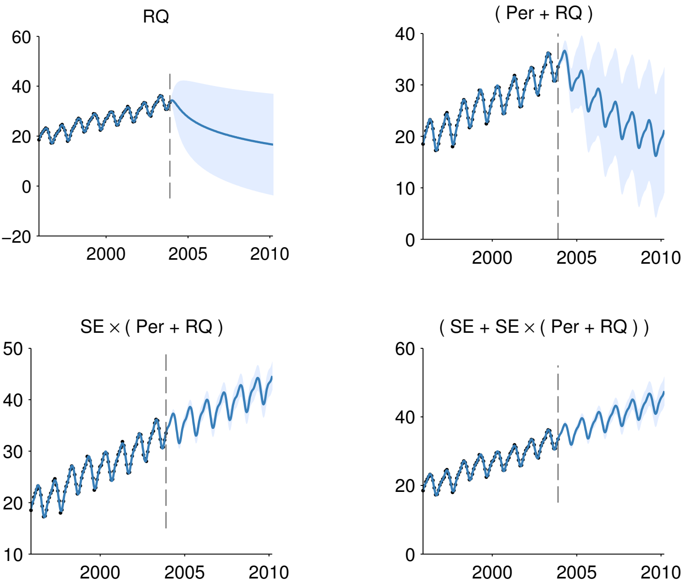
# Compositional Structure Search

- Define grammar over kernels:
  - $K \rightarrow K + K$
  - $K \rightarrow K \times K$
  - $K \rightarrow \{\mathrm{SE}, \mathrm{RQ}, \textsc{Lin}, \textsc{Per}\}$
- Search the space of kernels greedily by applying production rules, checking model fit (approximate marginal likelihood).

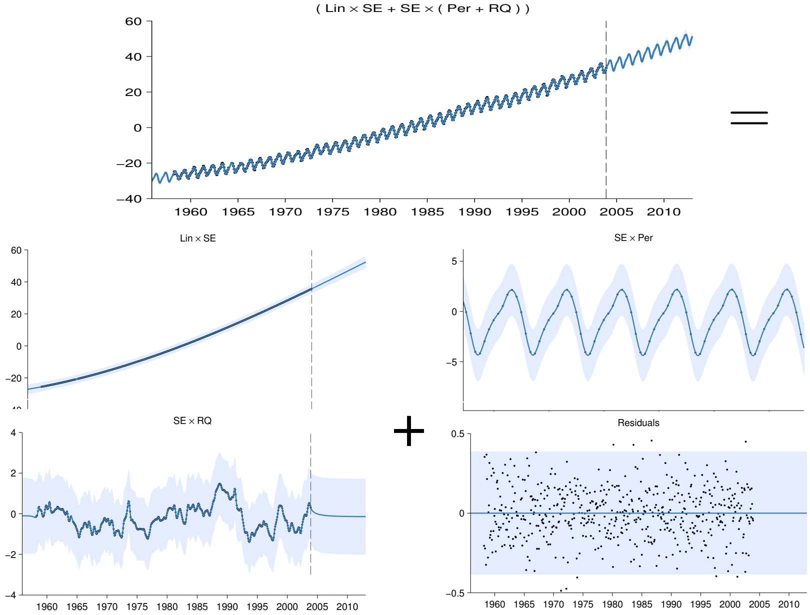
# COMPOSITIONAL STRUCTURE SEARCH



# EXAMPLE SEARCH: MAUNA LUA $CO_2$



# Example Decomposition: Mauna Loa $CO_2$



# COMPOUND KERNELS ARE INTERPRETABLE

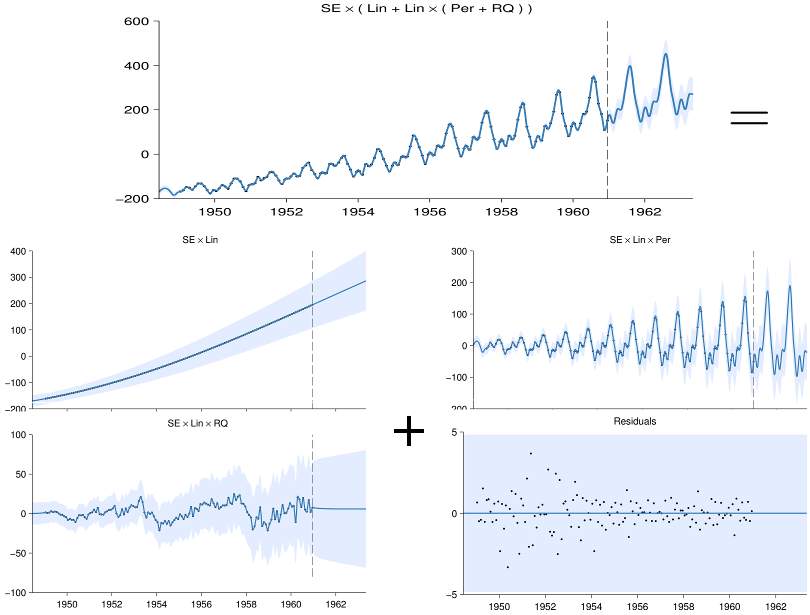
Suppose functions $f_1, f_2$ are draw from independent GP priors, $f_1 \sim \mathcal{GP}(\mu_1, k_1), f_2 \sim \mathcal{GP}(\mu_2, k_2)$. Then it follows that

$$f := f_1 + f_2 \sim \mathcal{GP}(\mu_1 + \mu_2, k_1 + k_2)$$

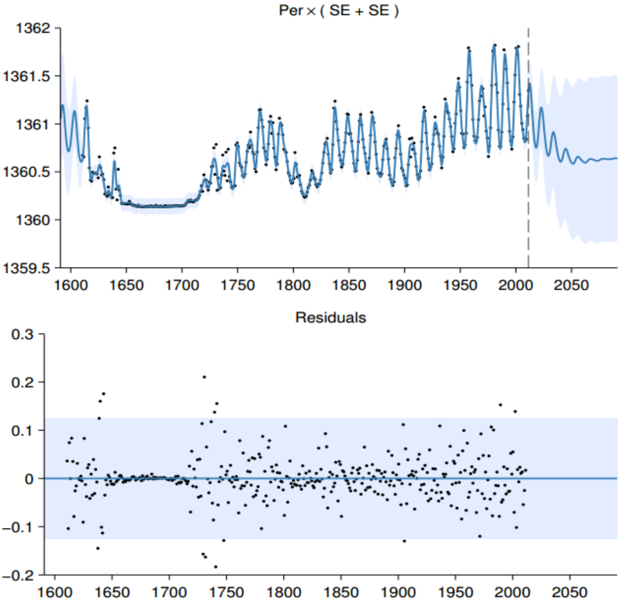
Sum of kernels is equivalent to sum of functions. Distributivity means we can write compound kernels as sums of products of base kernels:

$$\text{SE} \times (\text{RQ} + \text{LIN}) = \text{SE} \times \text{RQ} + \text{SE} \times \text{LIN}.$$

# EXAMPLE DECOMPOSITION: AIRLINE

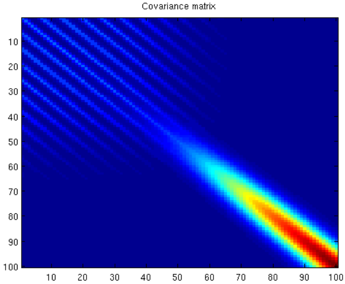


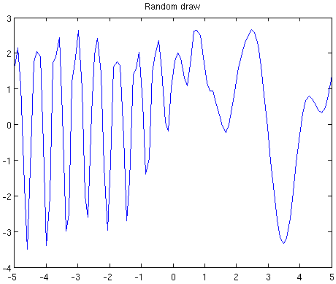
# EXAMPLE: SUNSPOTS



# Changepoint Kernel

Can express change in covariance:

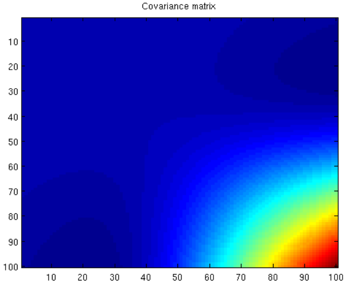


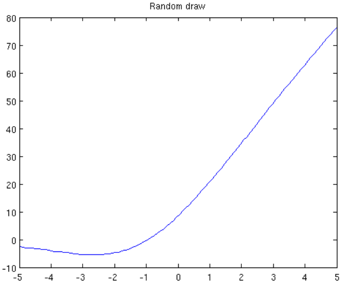




Periodic changing to SE

# CHANGEPOINT KERNEL

Can express change in covariance:




SE changing to linear

# Summary

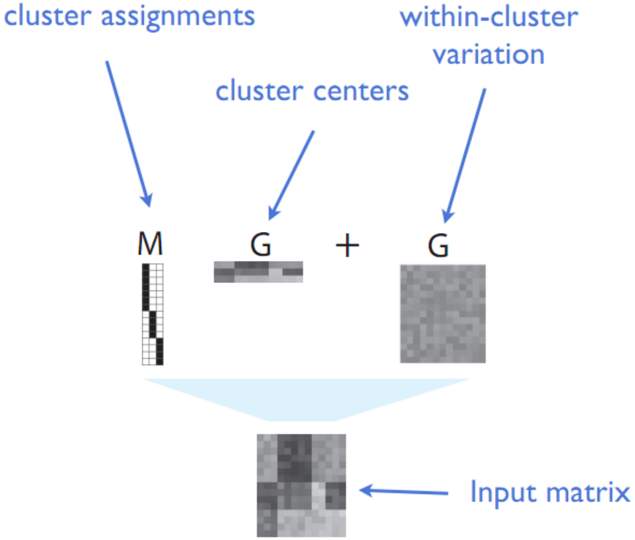
- Choosing form of kernel is currently done by hand.
- Compositions of kernels lead to more interesting priors on functions than typically considered.
- A simple grammar specifies all such compositions, and can be searched over automatically.
- Composite kernels lead to interpretable decompositions.

# Grammars for Matrix decompositions

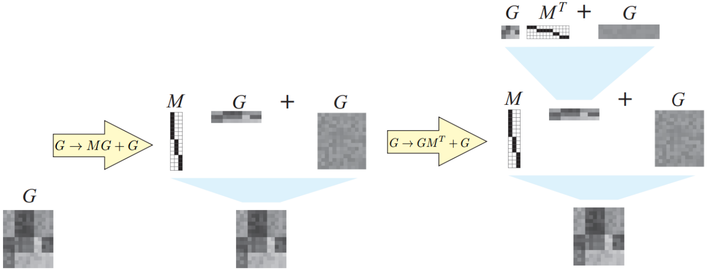
Previous work introduced idea of grammar of compositions:

- Exploiting compositionality to explore a large space of model structures [UAI 2012]
  Roger B. Grosse, Ruslan Salakhutdinov,
  William T. Freeman, Joshua B. Tenenbaum
- Slides that follow are from Roger Grosse

# Matrix decomposition



# Recursive Matrix decomposition



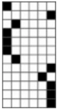

- Main idea: Matrices can be recursively decomposed
- Example: Co-clustering by clustering cluster assignments.

# Building Blocks

Gaussian
(G)

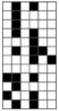
$$\lambda_i \sim \text{Gamma}(a, b)$$
$$\nu_j \sim \text{Gamma}(a, b)$$
$$u_{ij} \sim \text{Normal}(0, \lambda_i^{-1}\nu_j^{-1})^*$$

* variance parameters shared
between input rows/columns



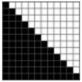
Multinomial
(M)

$$\pi \sim \text{Dirichlet}(\alpha)$$
$$u_i \sim \text{Multinomial}(\pi)$$



Bernoulli
(B)

$$p_j \sim \text{Beta}(\alpha, \beta)$$
$$u_{ij} \sim \text{Bernoulli}(p_j)$$

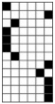


Integration
(C)

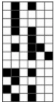
$$u_{ij} = \begin{cases} 1 & \text{if } i \geq j \\ 0 & \text{otherwise} \end{cases}$$

# MATRIX DECOMPOSITION: GRAMMAR

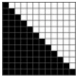
Gaussian (G)



Multinomial (M)



Bernoulli (B)



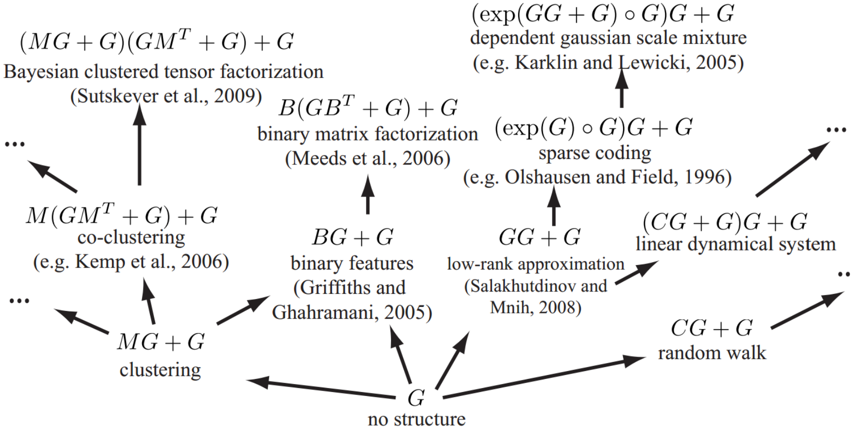
Integration (C)

**Starting symbol:** G

**Production rules:**

| | |
|---|---|
| clustering | $G \rightarrow MG + G \mid GM^T + G$ |
| | $M \rightarrow MG + G$ |
| low rank | $G \rightarrow GG + G$ |
| binary features | $G \rightarrow BG + G \mid GB^T + G$ |
| | $B \rightarrow BG + G$ |
| | $M \rightarrow B$ |
| linear dynamics | $G \rightarrow CG + G \mid GC^T + G$ |
| sparsity | $G \rightarrow \exp(G) \circ G$ |

# Matrix decomposition: Special cases



# Evolution of Image Models


Bossomaier and Snyder, 1987

Modeling images as linear combinations of uncorrelated basis functions gives a Fourier representation.

Modeling the sparse distribution of the linear reconstruction coefficients gives oriented edges.


Olshausen and Field, 1996


Karklin and Lewicki, 2005

Modeling the dependencies in the sparsity pattern gives a high-level texture model.

# Application to Natural Image Patches

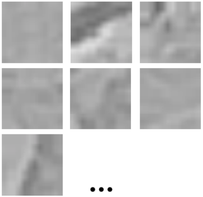
Image patches $GG + G$ $(\exp(G) \circ G)G + G$ $(\exp(GG + G) \circ G)G + G$

Model 1: Low-rank approximation (PCA).

Model 2: Sparsify coefficients to get sparse coding

Model 3: Model dependencies between scale variables

Data: 1,000 12x12 patches from 10 blurred and whitened images.



...

# CONCLUSIONS

- Model-building is currently done mostly by hand.
- Grammars over composite structures are a simple way to specify open-ended model classes.
- Composite structures often imply interpretable decompositions of the data.
- Searching over these model classes is a step towards automating statistical analysis.

# Conclusions

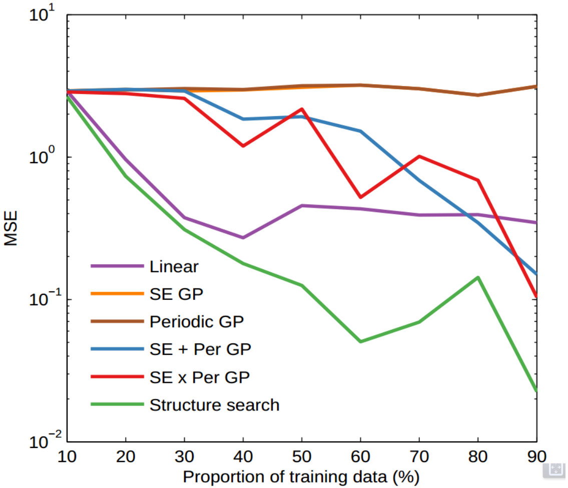
- Model-building is currently done mostly by hand.
- Grammars over composite structures are a simple way to specify open-ended model classes.
- Composite structures often imply interpretable decompositions of the data.
- Searching over these model classes is a step towards automating statistical analysis.

Thanks!

# Related Work

- Algorithmic information theory, e.g. Solomonoff induction (Solomonoff, 1964)
- Structure learning in other domains
  - Bayesian networks (e.g. Teyssier and Koller, 2005)
  - Markov random fields (e.g. Lee et al., 2006)
- Learning the form of graph embeddings (Kemp and Tenenbaum, 2008)
- Equation discovery
  - BACON knowledge discovery engine (Langley, Simon, and Bradshaw, 1984)
  - exploiting context-free grammar (Todorovski and Dzeroski, 1997)
- Matrix factorization frameworks
  - Exponential family PCA (Collins et al., 2002)
  - Roweis and Ghahramani (1999)
  - Singh and Gordon (2008)

# EXTRAPOLATION



# Multi-D Interpolation

| Method | Mean Squared Error (MSE) | | | | |
|---|---|---|---|---|---|
| | bach | concrete | puma | servo | housing |
| Linear Regression | 1.031 | 0.404 | 0.641 | 0.523 | 0.289 |
| GAM | 1.259 | 0.149 | 0.598 | 0.281 | 0.161 |
| HKL | **0.199** | 0.147 | 0.346 | 0.199 | 0.151 |
| GP SE-ARD | **0.045** | 0.157 | 0.317 | 0.126 | **0.092** |
| GP Additive | **0.045** | **0.089** | **0.316** | **0.110** | 0.102 |
| Structure Search | **0.044** | **0.087** | **0.315** | **0.102** | **0.082** |

# GRAMMARS FOR MATRIX DECOMPOSITIONS

1. **Gaussian (G).** Entries are independent Gaussians:

$$u_{ij} \sim \text{Gaussian}(0, \lambda_i^{-1}\lambda_j^{-1}).$$

This is our most generic component prior, and gives a way of deferring or ignoring structure.[1]

2. **Multinomial (M).** Rows are independent multinomials, with one 1 and the rest 0's:

$$\pi \sim \text{Dirichlet}(\alpha) \qquad u_i \sim \text{Multinomial}(\pi).$$

This is useful for clustering models, where $u_i$ determines the cluster assignment for the $i^{th}$ row.

3. **Bernoulli (B).** Entries are independent Bernoullis:

$$\pi_j \sim \text{Beta}(a, b) \qquad u_{ij} \sim \text{Bernoulli}(\pi_j).$$

   This is useful for binary latent feature models.

4. **Integration matrix (C).** Entries below the diagonal are deterministically 1:

$$u_{ij} = \mathbf{1}_{i \geq j}.$$

   This is useful for modeling temporal structure, as multiplying by this matrix has the effect of cumulatively summing the rows. (Mnemonic: C for "cumulative.")