Lecture 8: Optimization-based Approaches

Introduction to Modern Brain-Computer Interface Design

Christian A. Kothe
SCCN, UCSD

Outline

1. Introduction
2. Going Beyond CSP
3. Large-Scale Machine Learning
4. Application to the Spectral Model
5. Application to ERPs
6. Learning ERP and Oscillatory Weights Simultaneously
7. Practical Remarks

Swartz Center for Computational Neuroscience

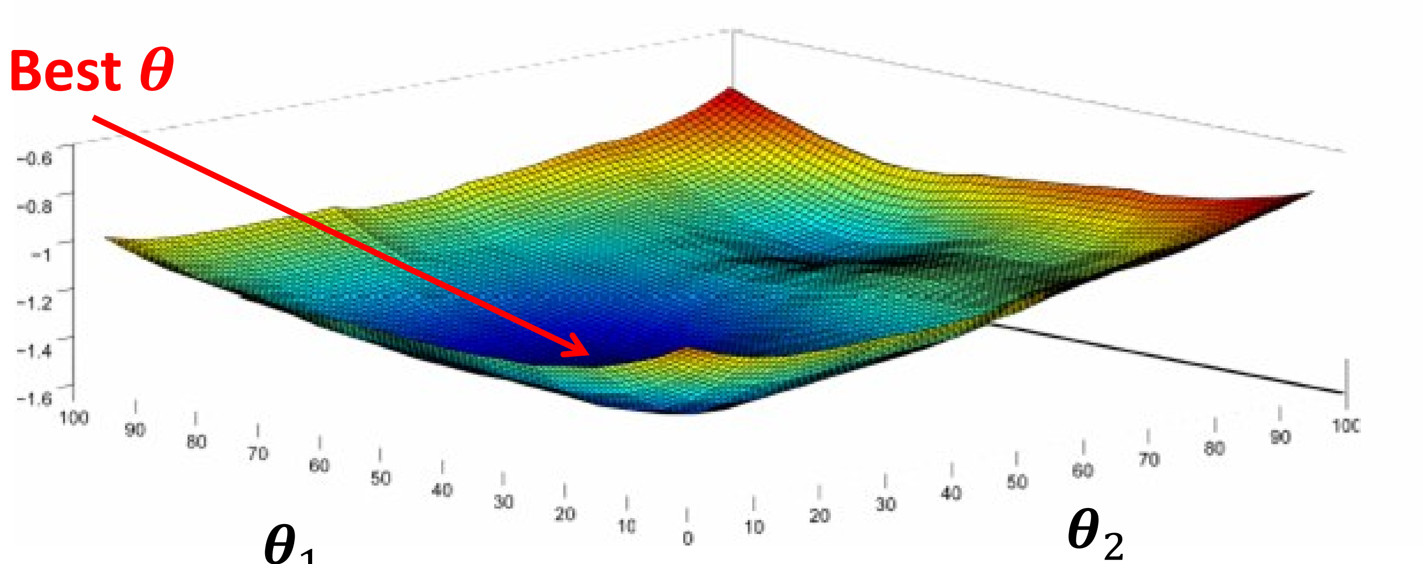
8.1 Introduction

Beyond CSP

- A unified, globally optimal solution to spatial filter estimation has recently been proposed (as an alternative to CSP+LDA)
- The method learns in a single step both the spatial filters and the relative weights for the filtered variance
- This is an *optimization*-based approach

Optimization

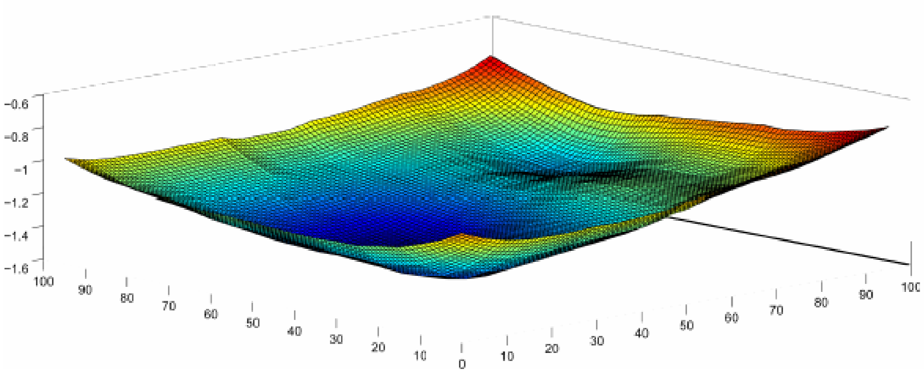
- Broad field, concerned with finding assignments to parameters that minimize a *cost function* $\boldsymbol{f}$
- Two branches: *Local optimization* and *Global Optimization*



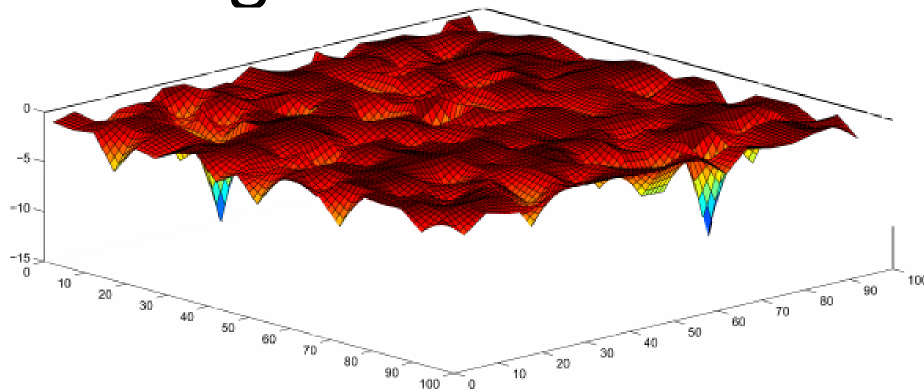
Swartz Center for Computational Neuroscience

Global Optimization

- Aims to find the global optimum of a function, even if it has multiple local optima
- Can be approximate (e.g., Simulated Annealing) or exact (e.g., Branch-and-Bound)
- **Problem:** Can be *extremely slow*, especially on high-dimensional and pathological data



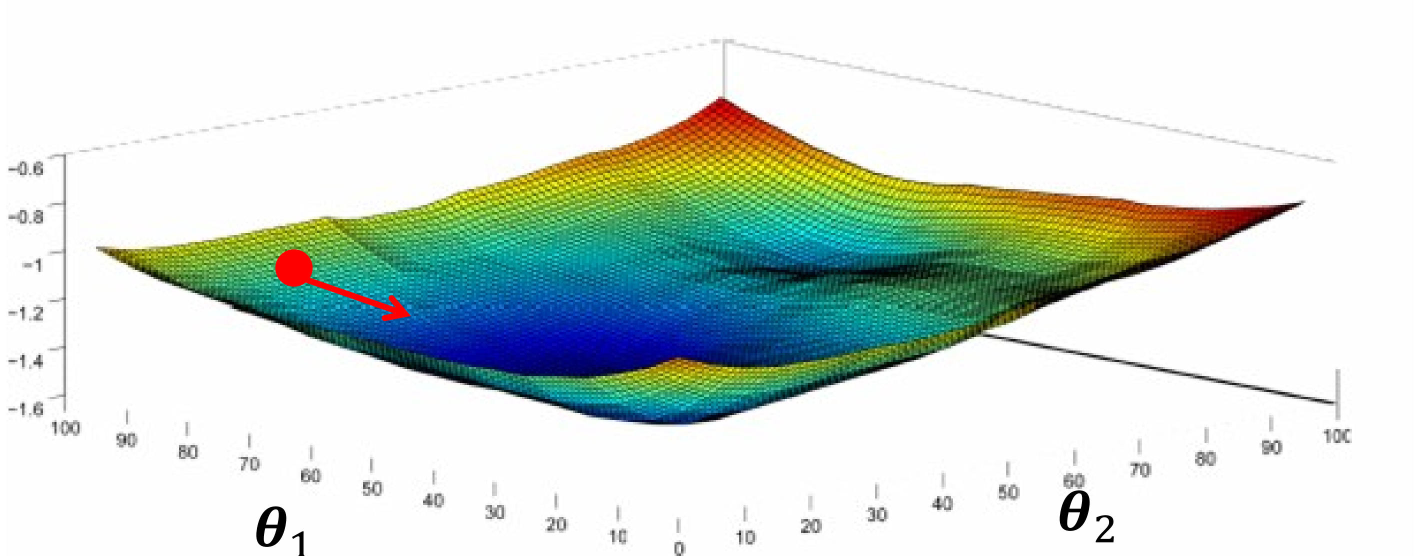
Doable



Problematic

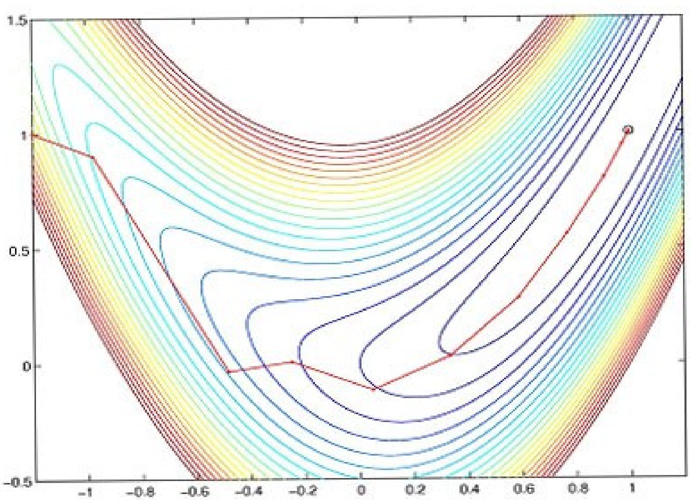
Local Optimization

- Improving an initial guess of a parameter by incremental updates
- **Method 1, Gradient Descent**: walk into the direction of steepest descent, requires 1st derivative $\boldsymbol{g}$ of cost function

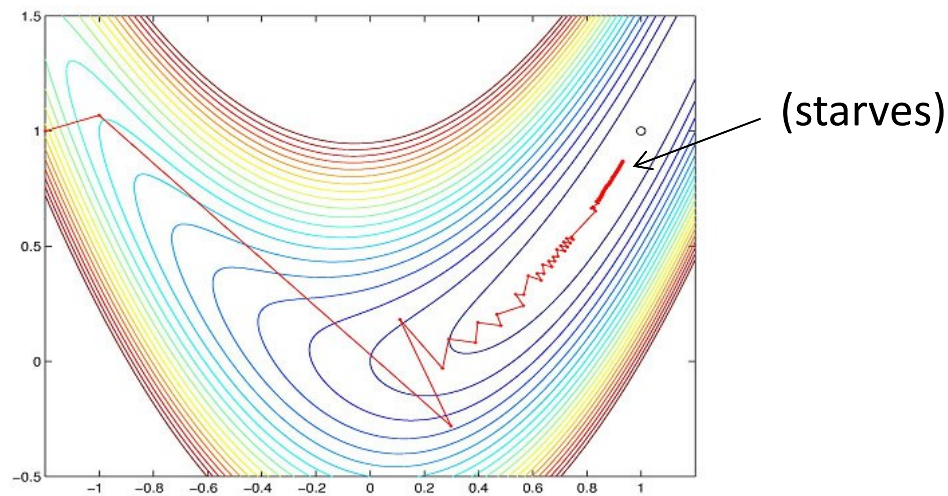


Local Optimization

- **Method 2, Newton Method**: account for local curvature, requires 2nd derivative $\boldsymbol{B}$ of function (Hessian matrix)
- Can be very efficient if $\boldsymbol{g}/\boldsymbol{B}$ is easy to compute (or easy to approximate otherwise)



Newton



Gradient Descent

Local Optimization

- **Method 3, Quasi-Newton**: approximates $\boldsymbol{B}^{-1}$ incrementally from gradients gathered along the way (no second derivatives necessary)
- **Practical variant:** L-BFGS (Limited-Memory Broyden-Fletcher-Goldfarb-Shanno) – one of the best known “off-the-shelf” second-order optimizers, *very easy* to use

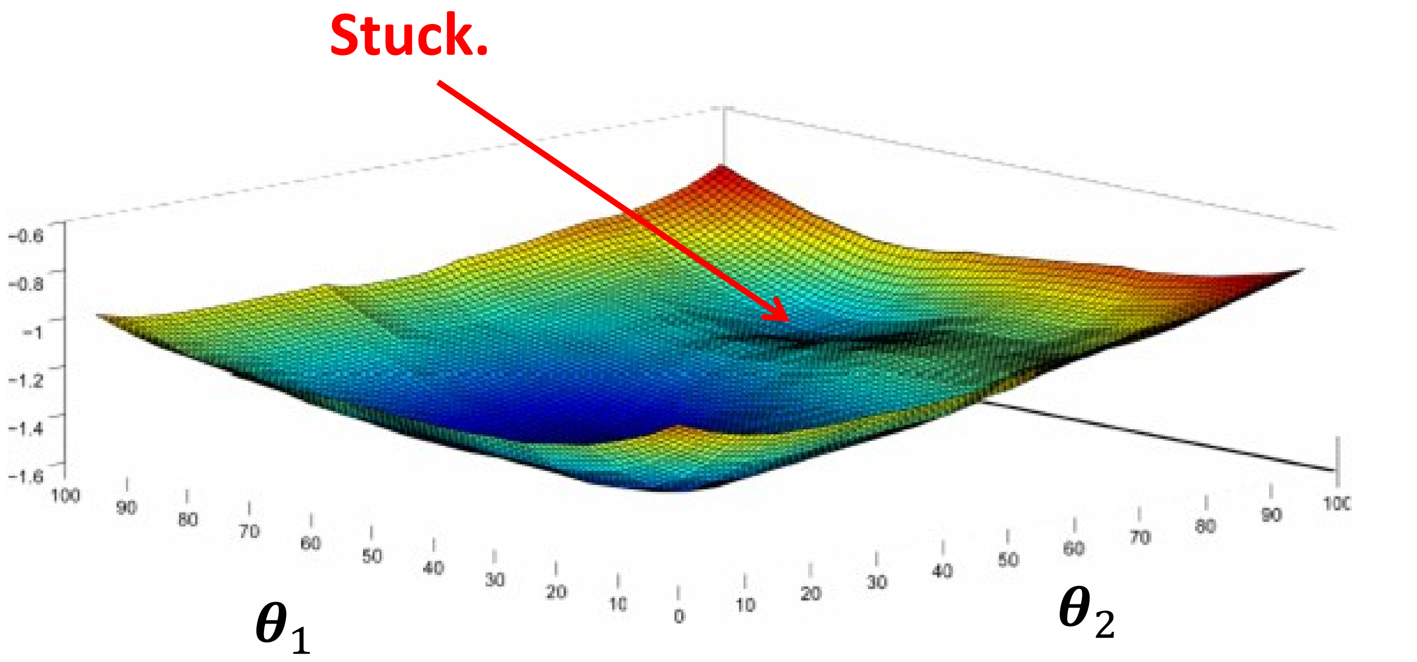
Local Optimization

- **Method 3, Quasi-Newton**: approximates $\boldsymbol{B}^{-1}$ incrementally from gradients gathered along the way (no second derivatives necessary)
- **Practical variant:** L-BFGS (Limited-Memory Broyden-Fletcher-Goldfarb-Shanno) – one of the best known “off-the-shelf” second-order optimizers, *very easy* to use

```
>> xopt = liblbfgs(@myfunc,x0)
```

Problems

- Local Optimization can run into *local minima* or *starve on plateaus*, except if the function has some suitable properties…



Convexity

- Convex functions have exactly one local optimum, also for every $\boldsymbol{x}$, $\boldsymbol{y}$ and $t \in [0,1]$ the following holds:

$$f(t\boldsymbol{x} + (1-t)\boldsymbol{y}) \leq tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y})$$

- This makes them *exactly solvable* by local optimization methods
- Surprisingly many problem types can be formulated as *convex optimization problems*

Smoothness

- Many relevant BCI problems amount to optimization problems with non-differentiable features (e.g., sparse problems), so smooth optimization is not immediately applicable
- **Fixes:** Can use smooth surrogate functions (e.g., *proximal optimization*), or solve an equivalent problem that is smooth (see *convex duality*) or split the non-smooth terms off (see *operator splitting*), or use non-smooth methods (e.g., *subgradient descent*)

8.2 Going Beyond CSP

(and a bit of equation juggling!)

Transforming CSP

- Consideration: Given a zero-mean trial $\boldsymbol{X} \in \mathbb{R}^{C \times T}$ with covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{CxC}$, spatial filters $\boldsymbol{W} \in \mathbb{R}^{SxC}$, linear weights $\boldsymbol{\theta} \in \mathbb{R}^{S}$ and bias b

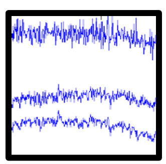
Transforming CSP

- Consideration: Given a zero-mean trial $\boldsymbol{X} \in \mathbb{R}^{C \times T}$ with covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{CxC}$, spatial filters $\boldsymbol{W} \in \mathbb{R}^{SxC}$, linear weights $\boldsymbol{\theta} \in \mathbb{R}^{S}$ and bias b

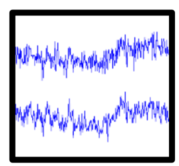
- Omitting the log from CSP, we have:

$$y = b + \boldsymbol{\theta}\text{var}(\boldsymbol{W}\boldsymbol{X})$$

X



WX



var(**WX**) $\begin{bmatrix} 2.3 \\ 4.5 \end{bmatrix}$

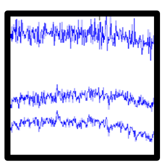
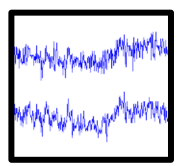
Swartz Center for Computational Neuroscience

Transforming CSP

- Consideration: Given a zero-mean trial $\boldsymbol{X} \in \mathbb{R}^{C \times T}$ with covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{CxC}$, spatial filters $\boldsymbol{W} \in \mathbb{R}^{SxC}$, linear weights $\boldsymbol{\theta} \in \mathbb{R}^{S}$and bias b

- Omitting the log from CSP, we have:

$$y = b + \boldsymbol{\theta}\mathrm{var}(\boldsymbol{W}\boldsymbol{X})$$

X  **WX**  var(**WX**) $\begin{bmatrix} 2.3 \\ 4.5 \end{bmatrix}$

- Rewriting in terms of individual spatial filters $\boldsymbol{W}_k$:

$$y = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \,\mathrm{var}(\boldsymbol{W}_k\boldsymbol{X})$$

Transforming CSP

- The variance term can be expressed using the covariance matrix $\boldsymbol{\Sigma}$ of segment $\boldsymbol{X}$:

$$y = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \operatorname{var}(\boldsymbol{W}_k \boldsymbol{X}) = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \left(\boldsymbol{W}_k \boldsymbol{\Sigma} \boldsymbol{W}_k{}^{\top}\right)$$

Transforming CSP

- The variance term can be expressed using the covariance matrix $\mathbf{\Sigma}$ of segment $\boldsymbol{X}$:

$$y = \ b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \, \mathrm{var}(\boldsymbol{W}_k \boldsymbol{X}) = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \left(\boldsymbol{W}_k \mathbf{\Sigma} \boldsymbol{W}_k{}^{\intercal} \right)$$

- And $\boldsymbol{W}_k \mathbf{\Sigma} \boldsymbol{W}_k{}^{\intercal}$ can be replaced by the inner product between two matrices $\left\langle \boldsymbol{W}_k \boldsymbol{W}_k{}^{\intercal}, \mathbf{\Sigma} \right\rangle$

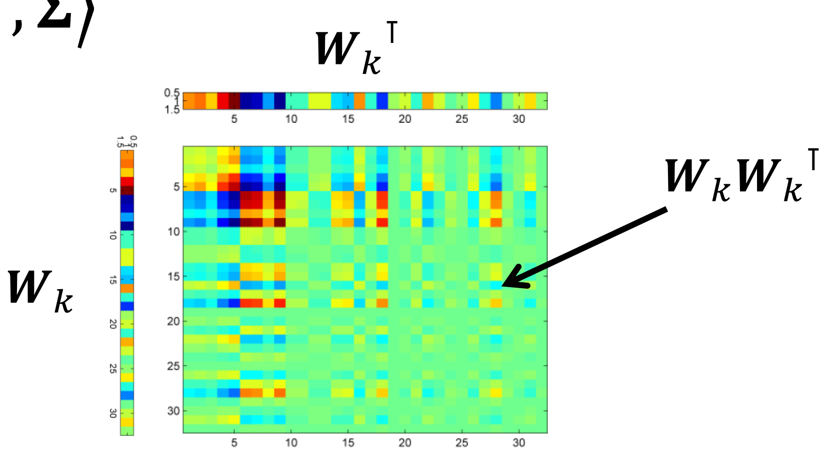
Transforming CSP

- The variance term can be expressed using the covariance matrix $\boldsymbol{\Sigma}$ of segment $\boldsymbol{X}$:

$$y = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \operatorname{var}(\boldsymbol{W}_k \boldsymbol{X}) = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \left(\boldsymbol{W}_k \boldsymbol{\Sigma} \boldsymbol{W}_k^{\top}\right)$$

- And $\boldsymbol{W}_k \boldsymbol{\Sigma} \boldsymbol{W}_k^{\top}$ can be replaced by the inner product between two matrices $\langle \boldsymbol{W}_k \boldsymbol{W}_k^{\top}, \boldsymbol{\Sigma} \rangle$

$$b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \langle \boldsymbol{W}_k \boldsymbol{W}_k^{\top}, \boldsymbol{\Sigma} \rangle$$



Transforming CSP

- The variance term can be expressed using the covariance matrix $\boldsymbol{\Sigma}$ of segment $\boldsymbol{X}$:

$$y = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \operatorname{var}(\boldsymbol{W}_k \boldsymbol{X}) = b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \left(\boldsymbol{W}_k \boldsymbol{\Sigma} \boldsymbol{W}_k^{\top}\right)$$

- And $\boldsymbol{W}_k \boldsymbol{\Sigma} \boldsymbol{W}_k^{\top}$ can be replaced by the inner product between two matrices $\langle \boldsymbol{W}_k \boldsymbol{W}_k^{\top}, \boldsymbol{\Sigma} \rangle$, and regrouped:

$$b + \sum_{k=1}^{S} \boldsymbol{\theta}_k \langle \boldsymbol{W}_k \boldsymbol{W}_k^{\top}, \boldsymbol{\Sigma} \rangle = b + \left\langle \sum_{k=1}^{S} \boldsymbol{\theta}_k \boldsymbol{W}_k \boldsymbol{W}_k^{\top}, \boldsymbol{\Sigma} \right\rangle$$

$$= b + \langle \boldsymbol{\Theta}, \boldsymbol{\Sigma} \rangle$$

Transforming CSP

- Thus this form is *linear in the covariance matrix* of X:

$$y = b + \langle \boldsymbol{\Theta}, \boldsymbol{\Sigma} \rangle = \boldsymbol{b} + \widetilde{\boldsymbol{\theta}}\mathrm{vec}(\boldsymbol{\Sigma})$$

- Could again learn $\widetilde{\boldsymbol{\theta}}$ using a simple linear method (e.g., LDA), but *very* high-dimensional (#parameters=$\mathrm{C}^{2^{2}}$)
- Need a method suitable for large-scale problems

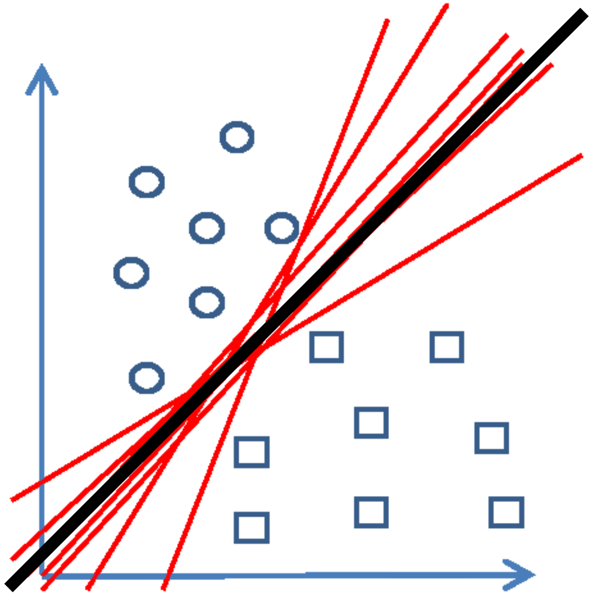
Swartz Center for Computational Neuroscience

8.3 Large-Scale Machine Learning

Swartz Center for Computational Neuroscience

Large-Scale Machine Learning

- *Discriminative* learning approaches like Support Vector Machines (SVMs) and Generalized Linear Models (GLMs) are well-adapted to high-dimensional / large-scale problems
- These directly optimize the parameters $\boldsymbol{\theta}$ given the data

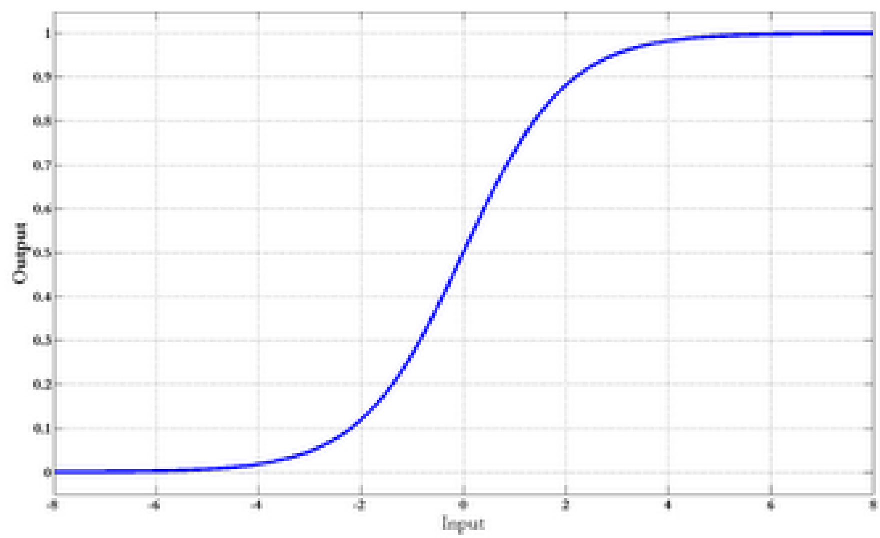


Swartz Center for Computational Neuroscience

Large-Scale Machine Learning

- Logistic Regression is a GLM that maps X onto binary outputs via a logistic "link function"

$$\frac{1}{1+e^{-y f_\theta(\boldsymbol{X})}}, (y \in \{-1,+1\})$$



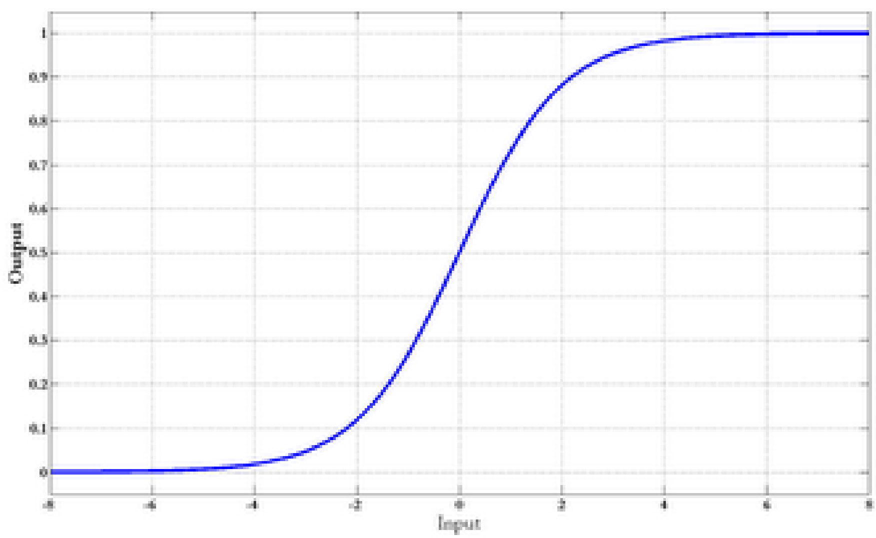
$- f_0(\boldsymbol{X}) \rightarrow$

Large-Scale Machine Learning

- Logistic Regression is a GLM that maps X onto binary outputs via a logistic "link function"

$$q_\theta(Y = y|X) = \frac{1}{1 + e^{-yf_\theta(X)}}, (y \in \{-1, +1\})$$

Interpreted as the probability that Y=1 (or Y=-1)



$- f_0(\boldsymbol{X}) \rightarrow$

Large-Scale Machine Learning

- Logistic Regression is a GLM that maps X onto binary outputs via a logistic "link function"

$$q_\theta(Y = y|X) = \frac{1}{1 + e^{-y f_\theta(\boldsymbol{X})}}, (y \ \in \{-1, +1\})$$

- … and linear function $f_\theta(\boldsymbol{X})$

$$f_\theta(\boldsymbol{X}) = \ \boldsymbol{\theta X} + b$$

Large-Scale Machine Learning

- $\boldsymbol{\theta}$ can be obtained via off-the-shelf convex optimization tools (such as CVX) by solving the problem

$$\min_{\boldsymbol{\theta}} \log\left(1 + e^{-\boldsymbol{y} f_{\theta}(\boldsymbol{X})}\right)$$

- The log(…) term is called the *logistic loss* and quantifies the misfit between predicted labels and true labels, for a particular choice of $\boldsymbol{\theta}$

Large-Scale Machine Learning

- For large problems, solution is still prone to *over-fitting* to random noise in the data – need to plug in some *additional assumptions*

$$\min_{\boldsymbol{\theta}} \log\left(1 + e^{-y f_{\theta}(\boldsymbol{X})}\right) + \lambda\Omega(\boldsymbol{\theta})$$

- Many choices for regularization term Ω
  - $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|^2$ encourages small weights
  - $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = |\boldsymbol{\theta}_1| + |\boldsymbol{\theta}_2| + \cdots$ encourages *sparsity*
  - can also get sparsity on groups of weights
  - combinations of these, …

8.4 Application to the Spectral Model

Applying to the Spectral Model

- In the previous supervised oscillatory model $y = b + \langle \boldsymbol{\Theta}, \boldsymbol{\Sigma} \rangle$, the matrix-shaped $\boldsymbol{\Theta}$ allows for a special matrix norm regularization $\Omega(\boldsymbol{\Theta})$:

$$\min_{\boldsymbol{\Theta}} \log\left(1 + e^{-y f_{\boldsymbol{\Theta}}(X)}\right) + \ \ldots$$

Applying to the Spectral Model

- In the previous supervised oscillatory model $y = b + \langle \boldsymbol{\Theta}, \boldsymbol{\Sigma} \rangle$, the matrix-shaped $\boldsymbol{\Theta}$ allows for a special matrix norm regularization $\Omega(\boldsymbol{\Theta})$:

$$\min_{\boldsymbol{\Theta}} \log\left(1 + e^{-y f_{\boldsymbol{\Theta}}(\boldsymbol{X})}\right) + \lambda \sum_{k=1}^{rank(\boldsymbol{\Theta})} \sigma_k(\boldsymbol{\Theta})$$

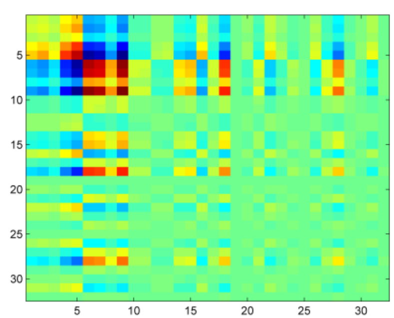
Applying to the Spectral Model

- In the previous supervised oscillatory model $y = b + \langle \boldsymbol{\Theta}, \boldsymbol{\Sigma} \rangle$, the matrix-shaped $\boldsymbol{\Theta}$ allows for a special matrix norm regularization $\Omega(\boldsymbol{\Theta})$:

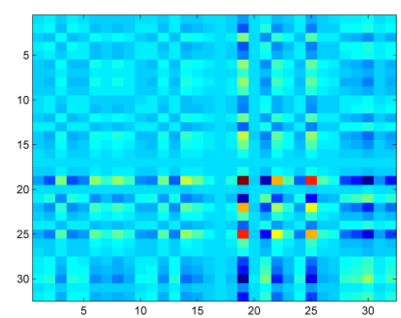
$$\min_{\boldsymbol{\Theta}} \log\left(1 + e^{-y f_{\boldsymbol{\Theta}}(\mathcal{X})}\right) + \lambda \sum_{k=1}^{rank(\boldsymbol{\Theta})} \sigma_k(\boldsymbol{\Theta})$$

- This encourages a low-rank structure in $\boldsymbol{\Theta}$, i.e.

$\boldsymbol{\Theta}$ =



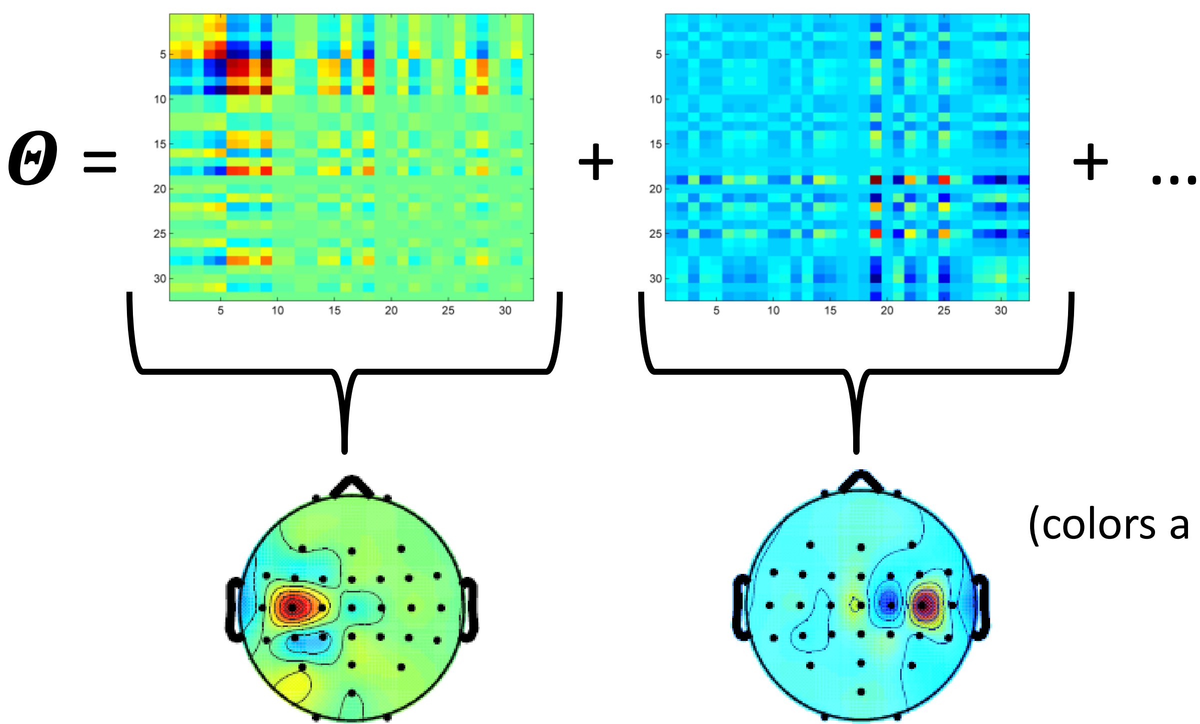
+



+ ...

Applying to the Spectral Model

- Thus, the weight matrix is equivalent to the weighted sum of a small set of spatial filters applied to the covariance matrix of the signal!



(colors a bit off)

Swartz Center for Computational Neuroscience

8.5 Application to ERPs

Application to ERPs

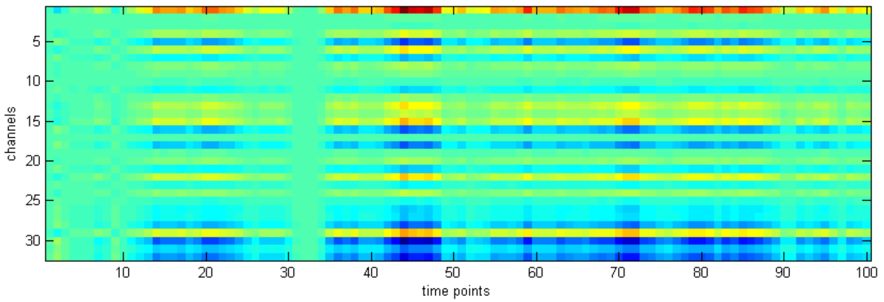
- Same approach can be applied to the raw data epoch **X** instead of its covariance matrix $\boldsymbol{\Sigma}$
- So we optimize for a GLM $y = b + \langle \boldsymbol{\Theta}, \boldsymbol{X} \rangle$

$$\min_{\boldsymbol{\Theta}} \log\left(1 + e^{-y f_{\boldsymbol{\Theta}}(\boldsymbol{X})}\right) + \lambda \sum_{k=1}^{rank(\boldsymbol{\Theta})} \sigma_k(\boldsymbol{\Theta})$$

Application to ERPs

- Same approach can be applied to the raw data epoch **X** instead of its covariance matrix $\boldsymbol{\Sigma}$
- This learns a linear ERP weight matrix with one weight for each channel and time point

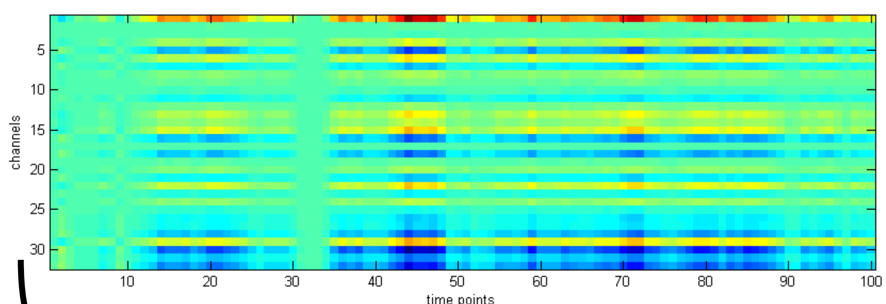
$\boldsymbol{\Theta}=$

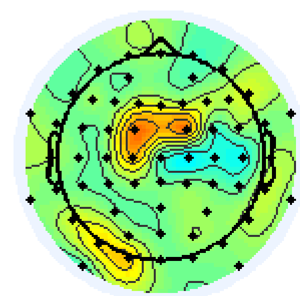


Application to ERPs

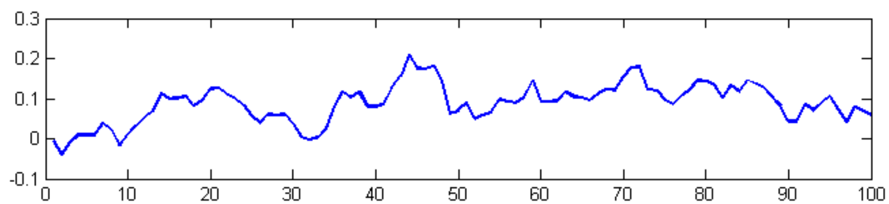
- Same approach can be applied to the raw data epoch **X** instead of its covariance matrix $\boldsymbol{\Sigma}$
- This learns a linear ERP weight matrix with one weight for each channel and time point

$\boldsymbol{\Theta}=$





spatial filter

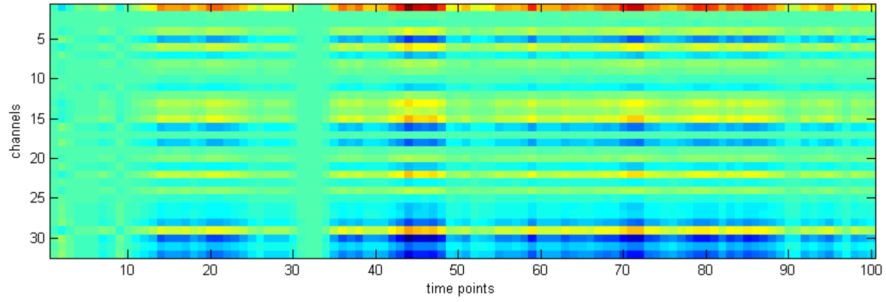


time weights

Application to ERPs

- Same approach can be applied to the raw data epoch $\mathbf{X}$ instead of its covariance matrix $\boldsymbol{\Sigma}$
- This learns a linear ERP weight matrix with one weight for each channel and time point

$\boldsymbol{\Theta}$ =



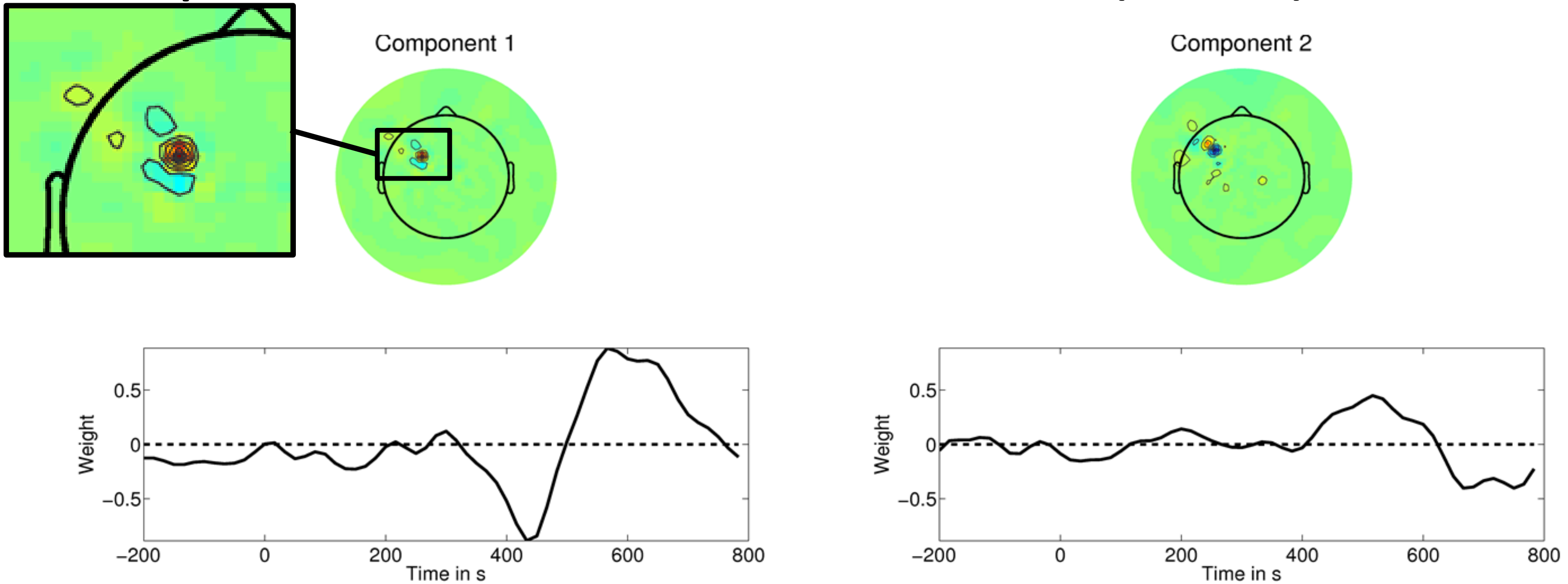
- $\boldsymbol{\Theta}$ is low-rank, so corresponds to a sum of a few spatial filters and their weights over time

Application to ERPs

- Thus, no hand-selected time windows needed
- Also, results are regularized to find / pick up few sources and their relevant time courses

Application to ERPs

- Thus, no hand-selected time windows needed
- Also, results are regularized to find / pick up few sources and their relevant time courses
- Rapid Serial Visual Presentation (RSVP) task:



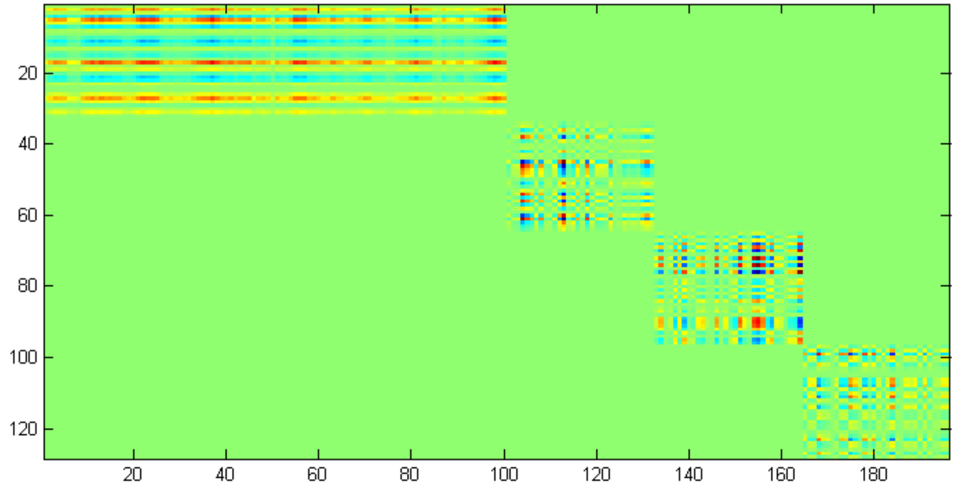
Swartz Center for Computational Neuroscience

8.6 Learning ERP and Oscillatory Weights Simultaneously

Feature Combination

- If for each trial instead of the covariance matrix or the raw ERP, a *block-diagonal concatenation* of both is used, the method learns a low-rank weight matrix combining both features simultaneously

$\boldsymbol{\Theta}$ =



Feature Combination

- If for each trial instead of the covariance matrix or the raw ERP, a *block-diagonal concatenation* of both is used, the method learns a low-rank weight matrix combining both features simultaneously
- For a block-diagonal weight matrix $\boldsymbol{\Theta}$, it holds that

$$\min_{\boldsymbol{\Theta}} \ldots + \lambda \sum_{k=1}^{rank(\boldsymbol{\Theta})} \sigma_k(\boldsymbol{\Theta})$$

is equivalent to solving for its blocks $\boldsymbol{\Theta}_b$:

$$\min_{\boldsymbol{\Theta}} \ldots + \lambda \sum_{b=1}^{B} \sum_{k=1}^{rank(\boldsymbol{\Theta}_b)} \sigma_k(\boldsymbol{\Theta}_b)$$

Sparsity and Feature Selection

- Also, covariance matrices for multiple frequency bands and time windows can be concatenated
- For regularizers that are a sum of terms (of a certain type), most terms in the sum will be driven to zero and only a *sparse subset* of terms remains non-zero, i.e., the relevant features are selected automatically

$$\min_{\boldsymbol{\Theta}} \ldots + \lambda \sum_{b=1} \ldots$$

- Recovery of the relevant support is statistically extremely efficient – it holds that the number of irrelevant dimensions under which the support can be accurately recovered is *exponential* in the number of observations (i.e., trials) [Ng 1998]

Sparsity and Feature Selection

- Thus, only the relevant subset of frequencies or time windows (for covariance) or ERP sources is typically learned
- Can be taken even further, e.g., could encourage weights for different time/frequency bins to share a small set of spatial filters (again using rank constraints on concatenated matrices) – this is called multi-task learning

Final Prediction Functions

- **Basic oscillatory case** (assuming **X** is band-passed):

$$y = \frac{1}{1 + e^{-\left(b + \langle \boldsymbol{\Theta}, \boldsymbol{X}\boldsymbol{X}^{\top} \rangle\right)}}$$

- **ERP case** (**X** can be band-passed)**:**

$$y = \frac{1}{1 + e^{-\left(b + \langle \boldsymbol{\Theta}, \boldsymbol{X} \rangle\right)}}$$

- **Combined cases** (here for temporal filters $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$):

$$y = \frac{1}{1 + e^{-\left\langle \boldsymbol{\Theta}, \begin{bmatrix} \boldsymbol{X} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X}\boldsymbol{F}_1\boldsymbol{X}^{\top} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{X}\boldsymbol{F}_2\boldsymbol{X}^{\top} \end{bmatrix} \right\rangle - b}}$$

Swartz
Center for
Computational
Neuroscience

8.7 Practical Remarks

Solving It

- Problems of this size are impossible to solve using CVX; need a custom solver
- For a wide range of sparse estimation problems the DAL (Dual-Augmented Lagrangian) solver is applicable and *very fast*
- For an even wider range of problems the ADMM (Alternating Direction Method of Multipliers) framework is applicable and also very fast

ADMM

- Framework for distributed very large scale optimization – leads to parallel algorithms
- Can be done with *very* simple MATLAB code

These are often fairly simple problems

$$
\begin{aligned}
x^{k+1} &:= \operatorname{argmin}_x L_\rho(x, z^k, y^k) \\
z^{k+1} &:= \operatorname{argmin}_z L_\rho(x^{k+1}, z, y^k) \\
y^{k+1} &:= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)
\end{aligned}
$$

Boyd, 2010

Other Methods in this Framework

- **Support Vector Machines:** use a different loss function (“hinge loss” instead of logistic loss)
- **Multiple Kernel Learning:** using group sparsity on kernel matrices (selecting few kernels)
- **Hierarchical Kernel Learning:** very advanced non-linear feature selection approach using tree-structured sparsity
- **Linear Regression:** Usable for a continuous output space instead of discrete

L8 Questions?