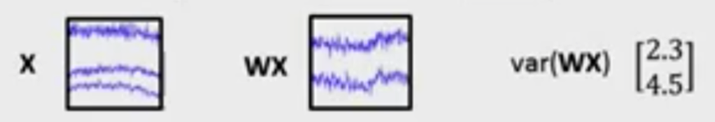
**Optimization-based Approaches**

1. Lecture 8.1 – Introduction
   1. Beyond CSP
      1. A unified, globally optimal solution to spatial filter estimation has recently been proposed (as an alternative to CSP+LDA)
         1. Applicable to spectral processes
         2. CSP+LDA is a two stage procedure which is hard to tell under which conditions is it optimal
      2. The method learns in a single step both the spatial filters and the relative weights for the filtered variance
         1. Single unified approach that gives you a globally optimal solution so you can prove so you can prove that what you get is optimal under certain assumptions
      3. This is an *optimization*-based approach
      4. Optimal for any stationary oscillatory process
   2. Optimization
      1. Broad field, concerned with finding assignments to parameters that minimize a *cost function* ***f***
         1. Starting with a cost function ***f*** let’s say and it has some parameters such as θ1 and θ2 and it quantifies the cost of using these parameters.
         2. The mismatch between what the prediction of a model for a certain set of parameters and what the calibration data says
      2. Two branches: *Local Optimization* and *Global*
   3. Global Optimization
      1. Aims to find the global optimum of a function, even if it has multiple local optima
      2. Can be approximate (e.g., Simulated Annealing) or exact (e.g., Branch-and-Bound)
      3. Problem: Can be *extremely slow*, especially on high-dimensional and pathological data
   4. Local Optimization
      1. Improving an initial guess of a parameter by incremental updates
      2. **Method 1, Gradient Descent**: walk into the direction of the steepest descent, requires 1st derivative ***g*** of cost function.
         1. Take an initial guess and improve it incrementally until you have found a local optimum
         2. It can get stuck on a plateau
      3. **Method 2, Newton Method**: account for local curvature, requires 2nd derivative ***B*** of function (Hessian matrix)
         1. Can be very efficient if ***g***/***B*** is easy to computer (or easy to approximate otherwise)
         2. Will shoot right to the surface versus Gradient Descent
      4. **Method 3, Quasi-Newton**: Approximates B-1 incrementally from gradients gathered along the way (no second derivatives necessary)
         1. The best of both method 1 & 2
         2. Approximates the Hessian matrix from the gradients that you say in previous iterations
         3. Don’t have to derive the equation for g/B
      5. **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
         1. Implements Method 3 paradigm  
            >> xopt = liblbfgs(@myfunc, x0)
   5. Problems
      1. Local optimization can run unto *local minima* or *starve on plateaus*, except if the function has some suitable properties
   6. Convexity
      1. Convex functions have exactly one local optimum, also for every ***x***, ***y*** and *t* ∈ [0,1] the following holds
      2. This makes them *exactly solvable* by local optimization methods
      3. Surprisingly many problem types can be formulated as *convex optimization problems*
      4. There are many applications of this optimization although it may not be easily visible
      5. If you are able to do this, any optimization technique will work so long as the new curve is smooth
   7. Smoothness
      1. Many relevant BCI problems amount to optimization problems with non-differentiable features (e.g., spares problems), so smooth optimization is not immediately applicable
         1. Such as absolute value which is very sharp
      2. 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*)
         1. Smooth out the non-smooth parts
         2. Can split the smooth and the non-smooth curves and deal with them separately
         3. Subgradient descent approaches from one side and approaches from the other and finds the agreement interaction
2. Lecture 8.2 – Going Beyond CSP
   1. Transforming CSP
      1. Consideration: Given a zero-mean trial (where C is number of channels and T is number of time points) with covariance (number of channels by number of channels), spatial filters (number of sources, or spatially filtered components, by number of channels), linear weights (linear weights like what LDA tends to compute, spatially filtered parameters) and bias *b*.
      2. Omitting the log form CSP, we have:



* + 1. Rewriting in terms of individual spatial filters :
       1. S is the number of spatially filter components
    2. The variance term can be expressed using the covariance matrix Σ of segment X:  
       1. Using the same trick seen in the CSP lecture
    3. And can be replaced by the inner product between two matrices
       1. Taking the matrix and vector transpose can rewrite this inner product between matrices
          1. Multiplied out you get a matrix with rank 1
    4. Thus this form is *linear in the covariance matrix* of X:
    5. Could again learn using a simple linear method (e.g., LDA), but *very* high-dimensional (#parameters = )
       1. 1. Weight vector
    6. Need a method suitable for large-scale problems

1. Lecture 8.3 – Large-Scale Machine Learning
   1. Large-Scale Machine Learning
      1. Discriminative learning approaches like Support Vector Machines (SVMs) and Generalized Linear Models (GLMs) are well-adapted to high dimensional / large-scale problems
      2. There directly optimize the parameters ***θ*** given the data
         1. Trying to change the parameters of the mapping from the space onto the features
         2. We are not messing with the data
      3. Logistic Regression s a GLM that maps C onto binary outputs via a logistic “link function”

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

* + - 1. …and linear function
    1. ***θ*** can be obtained via off-the-shelf convex optimization tools (such as CVX) by solving the problem
    2. The log(…) term is called the logistic loss and quantifies the misfit between predicted labels and true labels, for a particular choice ***θ***
    3. For large problems, solution is still prone to *over-fitting* to random noise in the data – need to plug in some *additional assumptions*
       1. Many choices for regularization term Ω
          1. encourages small weights
          2. encourages *sparsity*

Drives most parameters towards zero

* + - * 1. Can also get sparsity on groups with weights
        2. Combination of these, …
        3. There are almost 20 different types of sparsity
      1. I think my parameters are smooth or simple, or you know there is a systematic bias that may not show up in the data but you know your parameters are governed by that bias in some way. You can make sure you are accounting for that in your final solution!
      2. Extra added cost
      3. Can use cross-validation and parameter searching on *λ* to find the right setting… try a bunch of values and see how well it works
  1. Taco

1. Lecture 8.4 – Application to the Spectral Model
   1. Apply to the Spectral Model
      1. In the previous supervised oscillatory model , the matrix-shaped ***Θ*** for a special matrix norm regularization :

This is a vector writing style

* + - 1. Worth noting that you can append the *b* onto the ***Θ*** matrix at the end, but sometime it is helpful to not keep it there because you don’t want to apply some penalties to it.
      2. However we need a penalty to reduce the chances of over-fitting the model  
         1. λ

A scalar regularization parameter

* + - * 1. σk

The kth singular value of your weight matrix

* + - * 1. rank(***Θ***)
    1. This encourages a low-rank structure in ***Θ***
       1. My singular value spectrum of the matrix is sparse you will end up with a matrix that has a few single values and has low rank. So you can write it as a sum of matrices with rank 1
    2. 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!!
       1. Because we are having this tunable parameter which tunes the complexity of the model in the sense we are able to learn a variable number of spatial filters and the right number is optimized by this cross-validation.
       2. If you search over the right lambda, you learn just as many spatial filters and therefore sources and that you can afford to fit using the data that you have.

1. Lecture 8.5 – Application to ERPs
   1. Application to ERPs
      1. Same approach can be applied to the raw data epoch X instead of its covariance matrix ***Σ***
      2. So we optimize for a GLM
      3. This learns a linear ERP weight matrix with one weight for each channel and time point
         1. Now the matrix assigns weights to time points and to channels. Because one trial is a matrix and we take the inner product of the weight vector and matrix
         2. For a rank 1 matrix the loading across channels is basically a spatial filter again, and the loading across time is the weights overtime for the spatial filter. So basically what we learn is that there is a source and how to pick it up, the source time course, and when it is relevant in time.
            1. For example that it is not that relevant in the beginning of the trial, but then you press a button and it becomes relevant. There maybe positivity and negativity.
      4. ***Θ*** is low-rank, so corresponds to a sum of a few spatial filters and their weights over time.
         1. Because it is low rank and not always rank 1, maybe rank 5 or so you are learning in effect a small number of sources and their associated time courses in this epoch, or the associated time weights for that, that are maximally predictive of your class label or your continuous output.
         2. Essentially you learn how many sources are relevant
      5. Thus, no hand-selected time windows needed
      6. Also, results are regularized to find / pick up few sources and their relevant time courses
      7. May take 20-30 minutes to optimize this for a good size of data
      8. Rapid Serial Visual Presentation (RSVP) task
2. Lecture 8.6 – Learning ERP and Oscillatory Weights Simultaneously
   1. The point of this whole section is to combine the ERP model and oscillatory processes model into one. The task now is to learn from both ERP features in the time domain and the oscillatory features at the same time, which of those features are relevant, what are the necessary spatial filters, what are the important sources.
   2. Feature Combination
      1. 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
         1. We are just concatenating all the matrices that we had before
         2. This is the kind of trial that we stick into the same optimization problem that we had before,
         3. Once concatenated into one matrix, we learn weight matrices that have the same shape, they will be zero everywhere except where there is potentially some ERP process happening in this matrix.
         4. They might look like spatial filters for some oscillatory processes.
      2. For a block-diagonal weight matrix Θ, it holds that   
         is equivalent to solving for its blocks
   3. Sparsity and Feature Selection
      1. Also, covariance matrices for multiple frequency bands and time windows can be concatenated
      2. For regularizes 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
      3. 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]
         1. There are statistically guarantees that have been derived which says that finding which ones of all these possible features are relevant, are non-zero, and which ones are zero can be done with very very high statistical efficiency.
         2. Basically you can find out which non-zero pattern is correct for some data for a number of irrelevant features its exponential in the number of observations you made, so if you have 100 trials you can learn which ones have the right features out of a million features.
            1. Super huge! Like what the heck! Wow.
            2. You can have millions of features and if you frame such that you only need a few of them that some how it is sparse in some sense
            3. Easy to implement
         3. Thus, only the relevant subset of frequencies or time windows (for covariance) or ERP sources is typically learned
         4. 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 multitask learning.
            1. Simple Example… You have multiple subjects and you are learning a weight matrix, and for one subject you learn one vector and for another subject you learn another vector and so on and you do this for a whole matrix, say ten subjects. If your solving all this as one optimization problem but it covers all these subjects you can effectively encourage that this whole matrix is low rank and there is a small number of latent factors in the sense of weight profiles replicated across subjects and each subject has a different loading on a small number of factors. So you have some kind of sharing of statistical strength across seemingly unrelated tasks.

Check out neuromatters… it’s a BCI company doing this

* 1. Final Prediction Functions
     1. **Basic Oscillatory Case** (assuming **X** is band-passed):  
        1. If you are calculating the probability that your label is (+1) instead of (-1) you could say why is this result, it is the logistic term and here is the spelled out linear map.
        2. is similar to the covariance matrix
     2. **ERP Case** (**X** can be band-passed):  
        1. Just the data itself, the X
     3. **Combined cases** (here for temporal filters ***F***1 and ***F***2):  
        1. It is a matrix that is the concatenation of these blocks
        2. X is just the data
        3. is just a covariance matrix with some temporal filter
        4. is just another covariance matrix with some temporal filter

1. Lecture 8.7 - Practical Remarks
   1. Solving It
      1. Problems of this size are impossible to solve using CVX; need a custom solver
      2. For a wide range of sparse estimation problems the DAL (Dual-Augmented Lagrangian) solver is applicable and *very fast*
         1. It was originally written to solve these kinds of complex problems
      3. For an even wider range of problems the ADMM (Alternating Direction Method of Multipliers) framework is applicable and also very fast
         1. Does not matter how many terms you have as long as they are all convex
         2. Gives rise to very simple MATLAB code
   2. ADMM
      1. Framework for distributed very large scale optimization – leads to parallel algorithms
      2. Can be done with very simple MATLAB code
      3. There is some paper on how this works that explains in detail but basically it boils down to solving optimization problems for each term separately, say you have two terms of one and then the other, in many cases they are very simple, in one case it is just a logistical class and the other one is just an L1 norm or so thing. The third step is to fix the discrepancy between your two solutions in a sense, you fix the residual.
      4. Then you just iterate this and you alternatively optimize the various terms in round robin… there is a whole paper on this
   3. Other Methods in the Framework
      1. **Support Vector Machines**
         1. Use a different loss function (“hinge loss” instead of logistic loss)
      2. **Multiple Kernel Learning**
         1. Using group sparsity on kernel matrices (selecting few kernels)
      3. **Hierarchical Kernel Learning**
         1. Very advance non-linear feature selection approach using tree-structure sparsity
         2. Very advance
         3. Learns small number of non-linear features basically
      4. **Linear Regression**
         1. Useable for continuous output space instead of discrete