Kutas Lab Cognitive Science UCSD

Marta Kutas
Distinguished Professor
Cognitive Science
UCSD

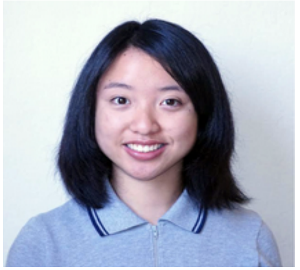
Tom Urbach
Project Scientist
Cognitive Science, UCSD

Qin Zhang
Bioinformatics Programmer
Cogntive Science, UCSD

Andrey Portnoy
Software Engineer
Cerebras Systems

Silas Horton
Programmer/Analyst
UCSD

Wen-hsuan Chan
Postdoctoral Researcher
Cognitive Science, UCSD



Lauren Liao
Graduate Student
Biostatistics, UC Berkely

Mentors

Suhas Somnath OLCF

Marty Kandes SDSC

Sneha Kottapalli NVIDIA

Max Katz NVIDIA

Ronnie Chaterjee OLCF

Julia Levites NVIDIA

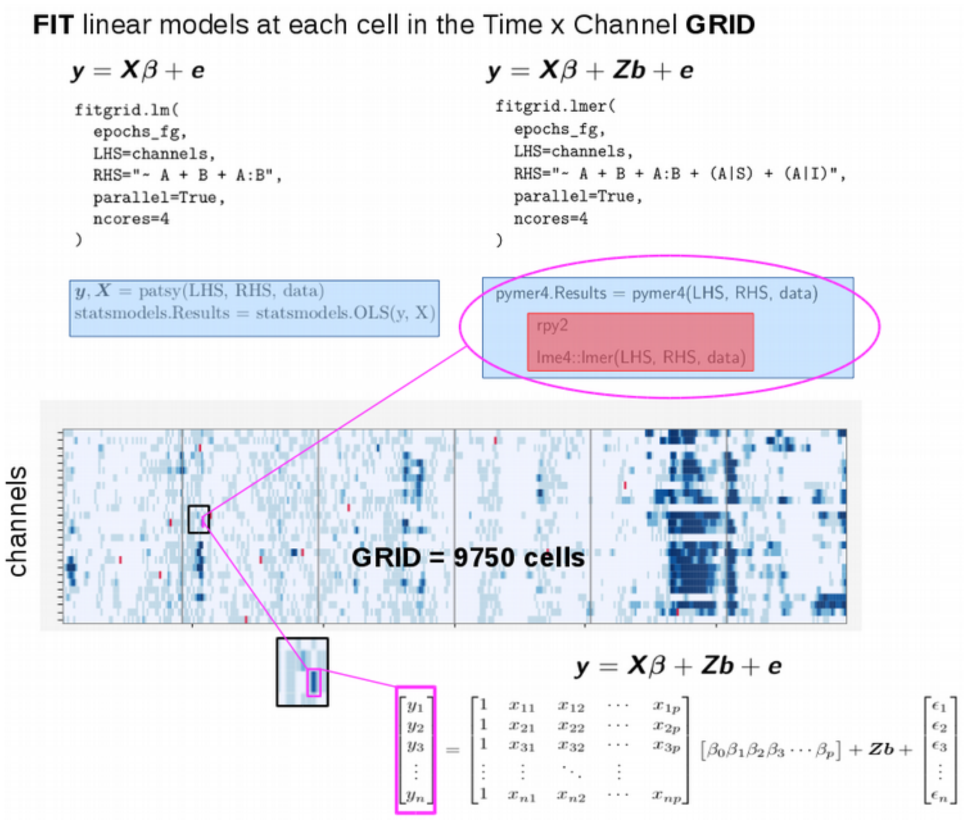
Mike Matheson ORNL

Junqi Yin ORNL

Our open-source app: **fitgrid** v0.4.10 Python, numpy, scipy, pandas, statsmodels, R, lme4

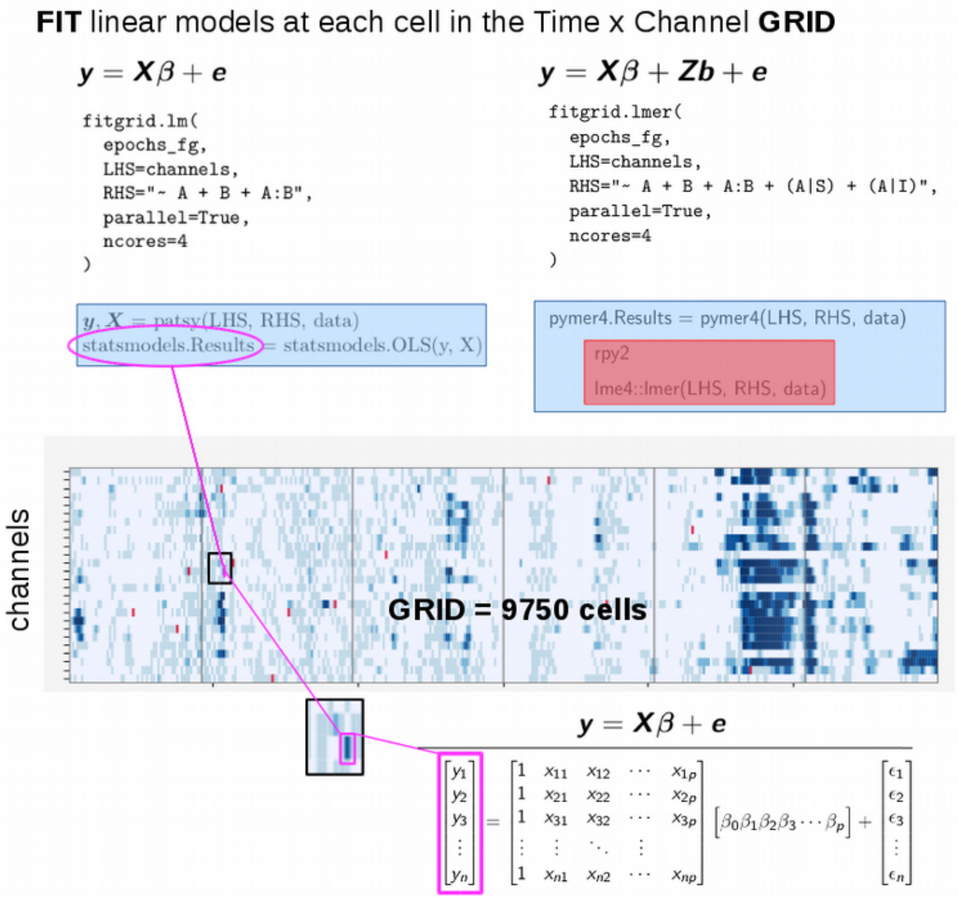
Hackathon Proposal

- Problem: sweep models across EEG
- Embarassingly parallel
- Target: LMM bottleneck
- Constraint: stay in Python



Pivot

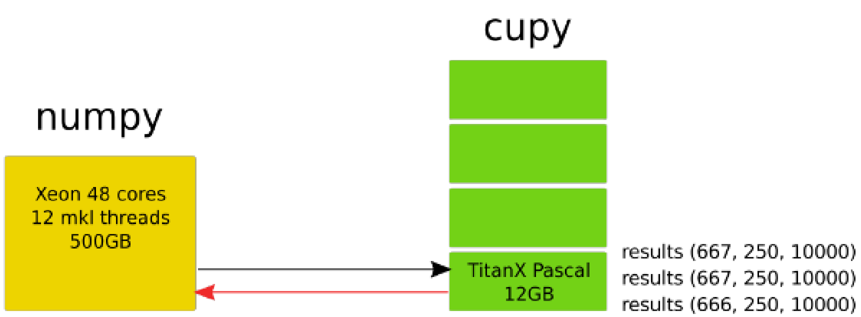
- LMM optimization not readily parallelizable
- New target: Large N LM bootstrap resampling

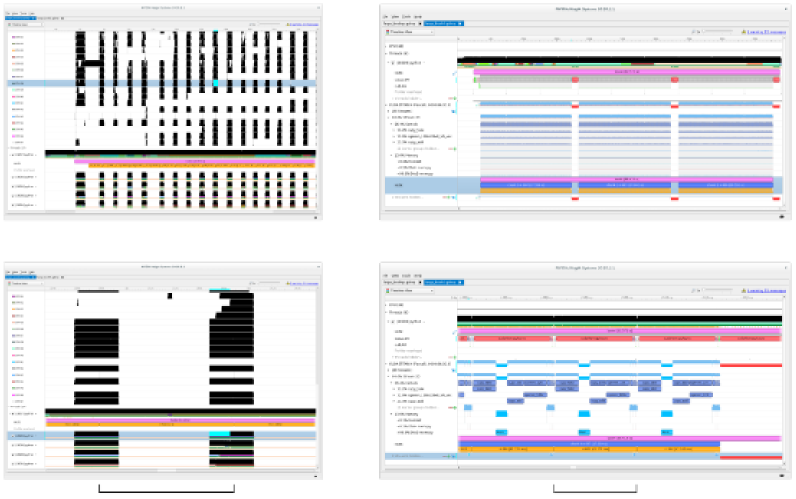


LM bootstrap resampling

25X

- numpy -> cupy
- no learning curve
- no new code
- no memory penalty



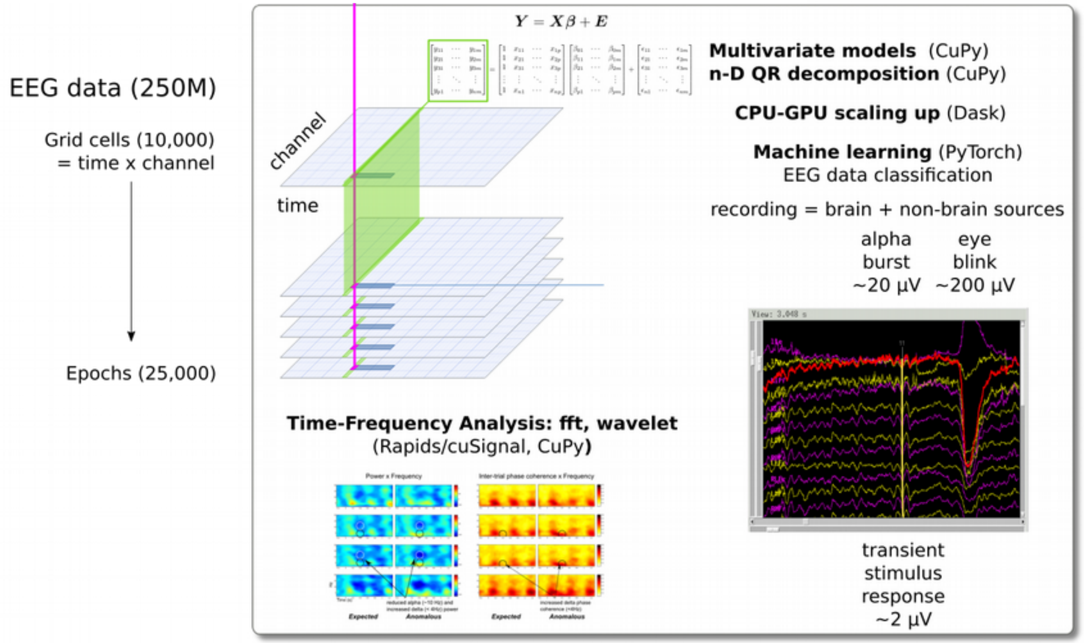


90 s = 80 iterations

90 s = 2000 iterations

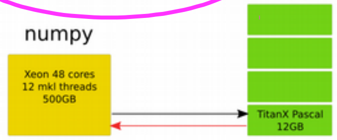
New Targets:

- Multivariate regression
- Frequency domain
- EEG classification



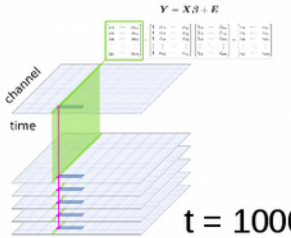
Multivariate Regression

6 - 14X



- R -> numpy -> cupy
- no learning curve
- 2 hr cupy API workaround (stride_tricks)

Y (25000, 256) = X (25000, 250) + E



t = 1000 sweeps

numpy in

```
# cupy computation
def get_mvbeta_cp(X, Y, t=100):
    X1 = cp.asarray(X)
    Y1 = cp.asarray(Y)
    n, p = X.shape
    n, m = Y.shape
    beta1 = cp.empty([t, p, m])
    for time_point in range(t):
        beta1[time_point, :, :] = cp.linalg.inv(X1.T @ X1) @ X1.T @ Y1
    b1 = beta1.get()
    return b1
```

cupy on

numpy out

results (1000, 250, 256)

vectorized multivariate regression QR solve

Time-frequency

- scipy.signal -> cuSignal
- no learning curve
- no recoding

our applications

5X - 20X

cuSignal benchmarks > 1000X

```
In [1]:
import cupy as cp
import cusignal
from scipy import signal
import numpy as np

cp.cuda.Device(3).use()
mempool = cp.get_default_memory_pool()
```

Cross Power Spectral Density

```
In [2]:
cx = np.random.rand(int(1e8))
cy = np.random.rand(int(1e8))
fs = int(1e6)

In [3]:
%%timeit
ccsd = signal.csd(cx, cy, fs, nperseg=1024)

7.37 s ± 932 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [4]:
mempool.free_all_blocks()

In [5]:
gx = cp.random.rand(int(1e8))
gy = cp.random.rand(int(1e8))
fs = int(1e6)

In [6]:
%%timeit
gcsd = cusignal.csd(gx, gy, fs, nperseg=1024);

The slowest run took 4.03 times longer than the fastest. This could mean that an intermediate result is being cached.
5.78 ms ± 2.79 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [7]:
mempool.free_all_blocks()
```

Periodogram

```
In [8]:
csig = np.random.rand(int(1e8))
fs = int(1e6)

In [9]:
%%timeit
f, Pxx_spec = signal.periodogram(csig, fs, 'flattop', scaling='spectrum')

18.3 s ± 122 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [10]:
gsig = cp.random.rand(int(1e8))
fs = int(1e6)

In [11]:
%%timeit
gf, gPxx_spec = cusignal.periodogram(gsig, fs, 'flattop', scaling='spectrum')

1.53 ms ± 84.6 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

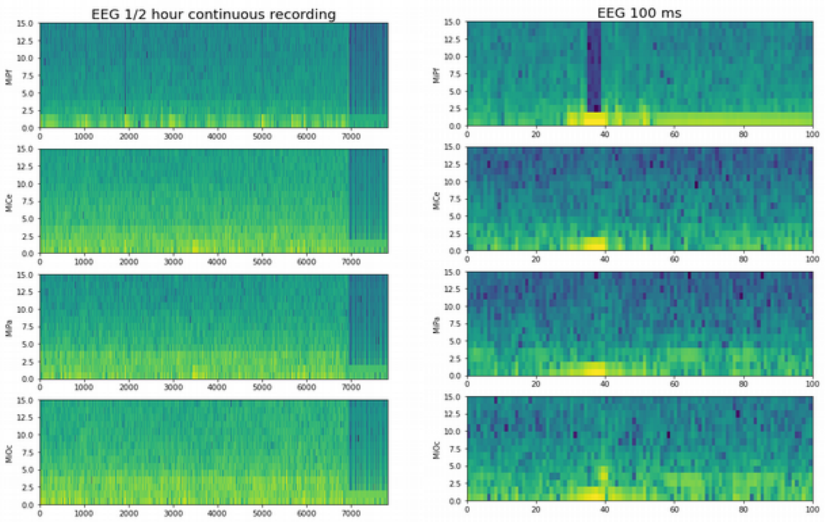
Short-time Fourier Transform (STFT)

Kutas Lab EEG data recording
1/2 hour x 32 channels

20 lines of Python
265 ms

```
%%timeit -n1 -r1

f, t, Zxx = stft_cp(eeg_df[EEG_CHANS].to_numpy(), 250, axis=0)
print(Zxx.shape)

freqs = slice(0, 15)
times = [
    ("EEG 1/2 hour continuous recording", slice(0, Zxx.shape[-1])),
    ("EEG 100 ms", slice(1900, 2000)),
]
for title, times in times:

    # short time FFT
    fig, axs = plt.subplots(4, 1, figsize=(8,12))
    for axi, chan in enumerate(["MiPf", "MiCe", "MiPa", "MiOc"]):
        ax = axs[axi]
        if axi == 0:
            ax.set_title(title, fontsize=18)
        ax.pcolormesh((np.log(np.abs(Zxx[freqs, CL[chan], times]))));
        ax.set(ylabel=chan)

(65, 32, 7833)
265 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

midline channels



1/2 hour

400 ms

EEG classification

Book work

April 2019

21595 downloads


Journal of Neural Engineering

TOPICAL REVIEW • OPEN ACCESS

Deep learning for electroencephalogram (EEG) classification tasks: a review

Alexander Craik[1], Yongtian He and Jose L Contreras-Vidal

Published 9 April 2019 • © 2019 IOP Publishing Ltd
Journal of Neural Engineering, Volume 16, Number 3

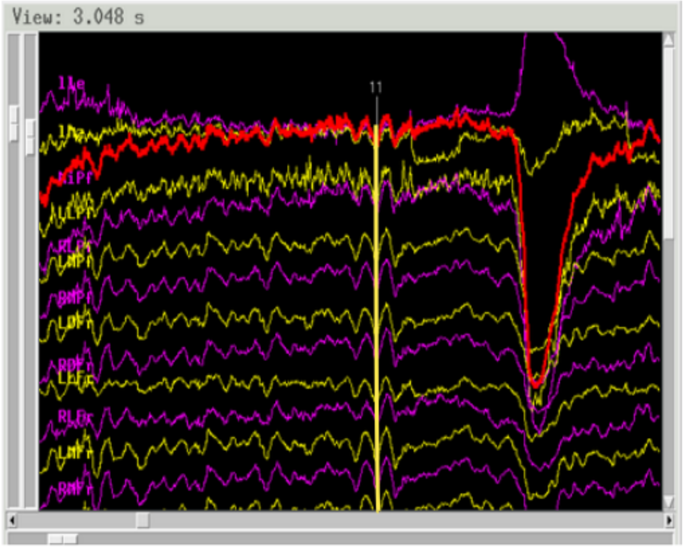
+ Article information

Abstract

Abstract

Objective. Electroencephalography (EEG) analysis has been an important tool in neuroscience with applications in neuroscience, neural engineering (e.g. Brain-computer interfaces, BCI's), and even commercial applications. Many of the analytical tools used in EEG studies have used machine learning to uncover relevant information for neural classification and neuroimaging. Recently, the availability of large EEG data sets and advances in machine learning have both led to the deployment of deep learning architectures, especially in the analysis of EEG signals and in understanding the information it may contain for brain functionality. The robust automatic classification of these signals is an important step towards making the use of EEG more practical in many applications and less reliant on trained professionals. Towards this goal, a systematic review of the literature on deep learning applications to EEG classification was performed to address the following critical questions: (1) Which EEG classification tasks have been explored with deep learning? (2) What input formulations have been used for training the deep networks? (3) Are there specific deep learning network structures suitable for specific types of tasks? *Approach*. A systematic literature review of EEG classification using deep learning was performed on Web of Science and PubMed databases, resulting in 90 identified studies. Those studies were analyzed based on type of task, EEG preprocessing methods, input type, and deep learning architecture. *Main results*. For EEG classification tasks, convolutional neural networks, recurrent neural networks, deep belief networks outperform stacked auto-encoders and multi-layer

Bench work



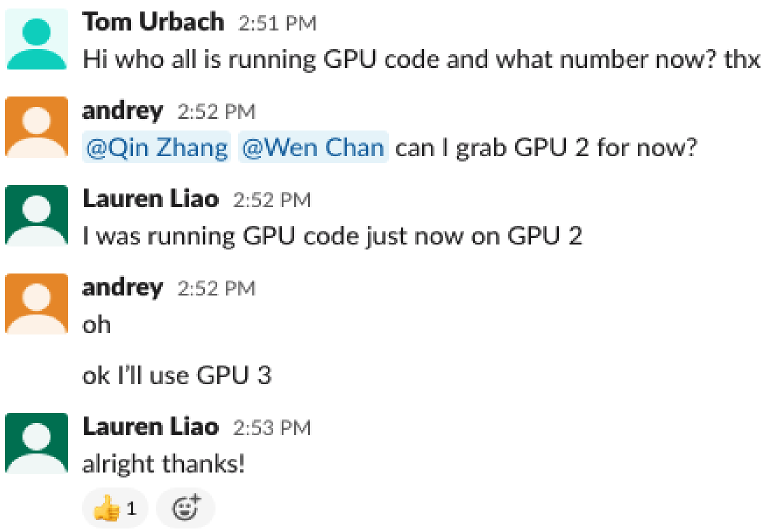
Best Practices

Progress

- GPU accelerated 3 of 5 targets without leaving Python
- 5X - 25X at realistic scales
- Scientific advance: randomization methods applicable at scale

Obstacles

- Usual environment, driver, package dependency wrangling
- 1 visit to CuPy Python github repo to look at source



Next Steps

- Scaling up, automagic resource distribution: Dask
- fine-tune GPU memory management

Thanks

- everyone for the interesting projects, problems, solutions
- Julia, Andi, Tom P., Sneha, mentors Suhas, Marty, Max
- Kutas Lab: Andrey, Qin, Wen, Lauren and Marta for the green light