

Real-Time MEG Analysis

Dr. Tom Holroyd
NIMH MEG Core Facility

Club Meg

20201023

Topics

- Hardware
- Examples
 - Subject Re-positioning
 - Neurofeedback
 - RT Beamforming
- Machine learning
 - Using beamformers and forward solutions
 - BCI

Hardware

- CTF 275-channel MEG



- Data from the SQUID electronics are sent in packets of about 90 samples (~75 ms at 1.2 kHz) to the Acquisition computer (called Squid)
- A Python script (acq_reader) gets copies:
 - Packets are converted to Python Numpy arrays and processed
 - Results can be sent over network connections to other computers
- Both serial and network connections are available for the stimulus computer (called Ika)
- A high-performance GPU-enabled workstation (called Kani) is available over the network (NFS / direct packets)

Hardware

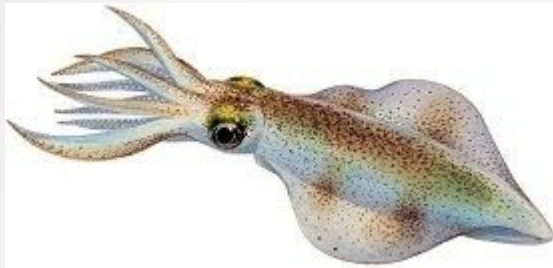
- Squid (aka the Acquisition computer)



- 8 core Intel Xeon CPU X5647 @ 2.93GHz
- 12 GB RAM
- Receives MEG data from CTF electronics
- Process and save/send to other computers

Hardware

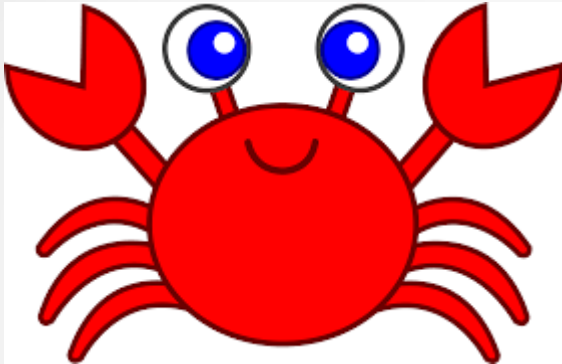
- Ika (Stimulus computer)



- Ika can talk to the Acq computer (Squid) by serial or TCP
- NVIDIA GTX 1060 GPU (6 GB)
- Example: The Head Repositioning System

Hardware

- Kani



- 20 core Intel Xeon W-2255 CPU @ 3.70GHz
- NVIDIA Quadro RTX 8000 GPU, 48 GB RAM
- 1 TB SSD & 4 TB HD storage
- Singularity container with Keras/Tensorflow 2.0

Subject Re-positioning

Continuous Head Localization Example

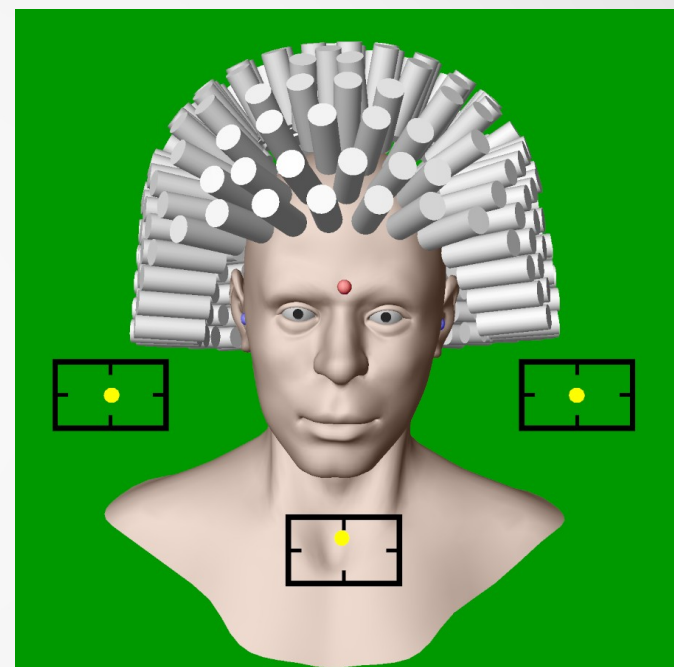
The CTF system can be configured to provide continuous output of head position information.

The `acq_reader` script extracts the head position information from the real-time data stream and outputs it to the stimulus computer over a serial link.

The stimulus computer displays a model of the subject's head position inside the MEG sensor.

Target locations for the fiducial coils, read from a previously recorded dataset, are displayed as boxes.

The subjects can use this to position themselves into a similar position to the one they were in for the previous recording.



This is what the subject sees

RT Neurofeedback (simple version)

We have an example script that does the following:

RT channel data are filtered into a narrow band (alpha) and an instantaneous power estimate is sent to the stimulus computer.

The stimulus computer displays this power level (which can in fact be a ratio of two different frequency band powers, or indeed anything) as the height of a ball over a stand.

The Acq computer is powerful enough to calculate a number of metrics to send to the stimulus computer for biofeedback purposes.

Virtual beamformer channels are also possible.

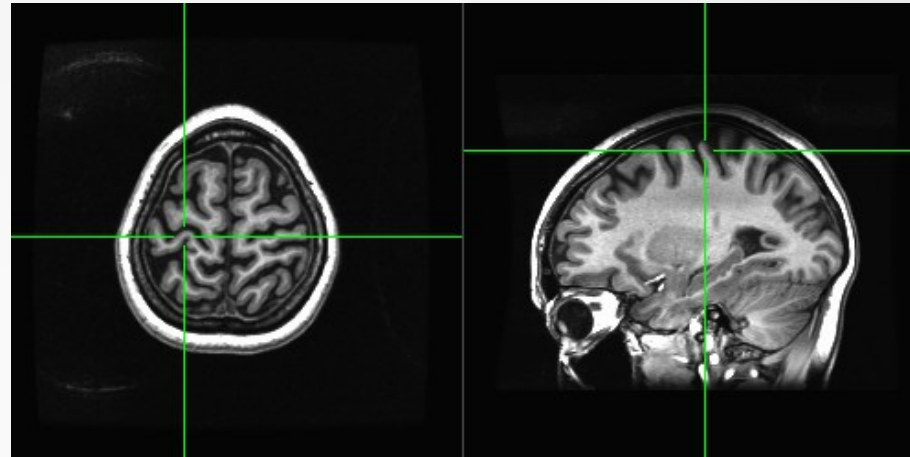


This is what the subject sees

RT Beamforming

Step one in creating a beamformer is calculating a head model, which requires an MRI.

The subject's MRI must be labeled with fiducial marks before the scan, and rotated into the correct coordinate system. Target coordinates can be selected beforehand.



Next, a baseline collection is required before RT operations, to establish the head position and covariance statistics. That dataset can then be used to establish a head model.

Beamformers are calculated on the Acq computer using the usual `sam_cov` and `sam_wts` programs. Beamformers are vectors, one per target source. Multiple beamformers can be used to calculate virtual timeseries in real time.

It's best to use very recent covariance data.

RT Beamforming

A beamformer is calculated from data, as well as a magnetic field calculation that depends on the head model.

Because the brain's statistics are non-stationary, the covariance matrix used to create the beamformer may change during a long recording.

To deal with this, we have an algorithm called SER (Widrow & Stearns) which calculates the covariance matrix on the fly. Data samples can be sent to a separate process on the Acq computer which continuously updates a covariance matrix.

This RT covariance can be used to update the beamformer, which improves S/N.

Machine Learning

- RT MEG data are sent over the network to the compute engine Kani, either via direct socket or NFS filesystem.
- Pre-computed beamformers can be loaded into a Deep Neural Network (DNN) model.
 - This allows the GPU to calculate virtual channels.
- Forward solutions can be computed (sam_wts -B) and used by the network to reconstruct the input.

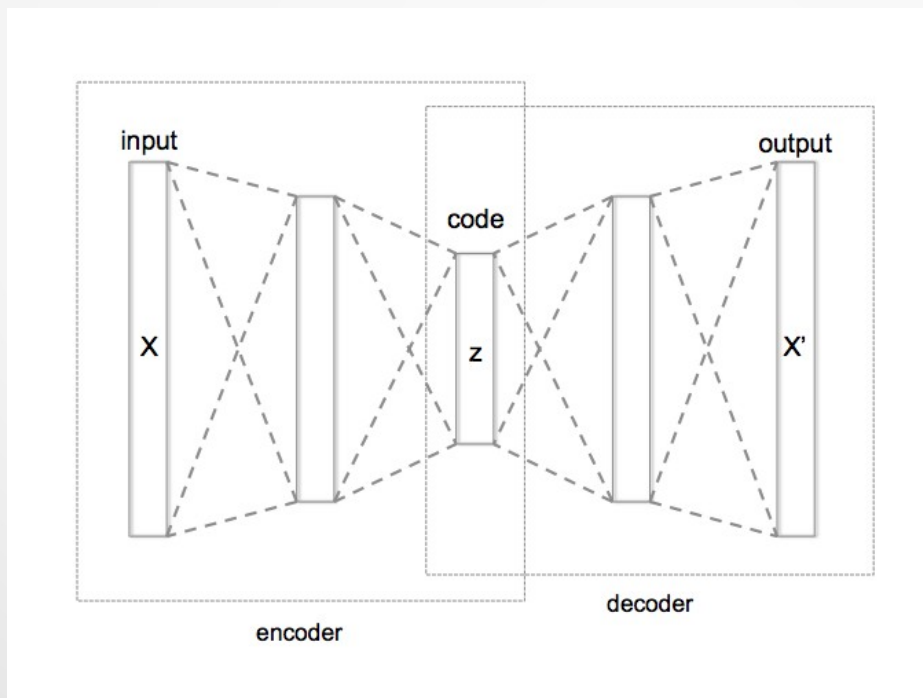
Machine Learning

- A beamformer is just a vector of 275 numbers, one per channel. A virtual channel is just the inner product of the filtered data with a beamformer. This is what a neural network does (linear combination).
- N beamformers can be loaded into a DNN tensor layer. Then the input 275-channel MEG data is automatically turned into N virtual channels inside the DNN.
- Forward solutions ($\text{sam_wts} \cdot B$), the computed magnetic fields from the head model, can be loaded into DNN layers too. Then the network can, using deeper, trained layers, compute what the magnetic field would look like.

Autoencoder

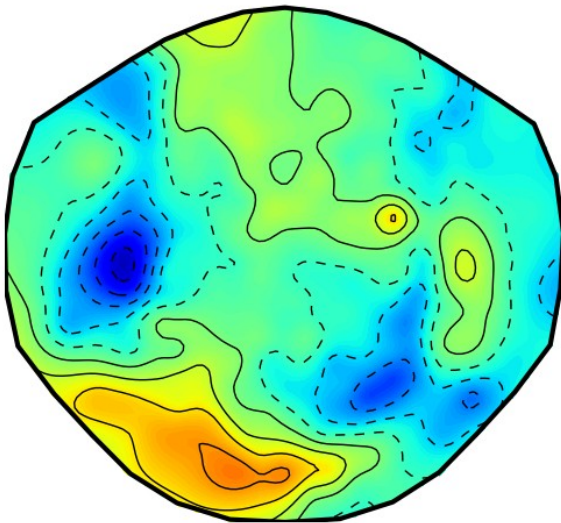
An autoencoder learns to reproduce its input.

That is, the output is trained on the input (self-supervised learning). The middle layer “code” is a lower dimensional representation of the input.

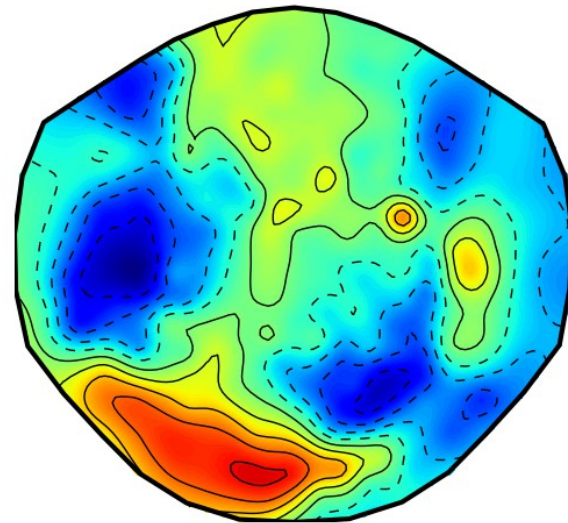


Autoencoder Input & Output

This network had 50 weights in the middle layer. Contrast has been enhanced due to elimination of unmodeled noise.



Raw MEG Data



Reconstruction

Classifier with Frozen Beamformers

Keras/Tensorflow code that loads beamformers into a frozen input layer.

```
input_tensor = Input(shape = (seqlen, M))  
l = Dense(Nbeam, activation = 'linear', bias = False)  
x = l(input_tensor)
```

```
l.set_weights([Beam])  
l.trainable = False
```



```
l1 = x = Lambda(lambda x: x * x)(x)
```

```
x = BatchNormalization()(x)  
x = Dropout(.5)(x)
```

```
x = Convolution1D(30, 15, activation = 'softplus')(x)  
x = Convolution1D(30, 15, activation = 'softplus')(x)  
x = MaxPooling1D(2)(x)
```

```
x = BatchNormalization()(x)
```

```
x = Dropout(.5)(x)
```

```
x = Convolution1D(30, 10, activation = 'softplus')(x)
```

```
x = Convolution1D(30, 10, activation = 'softplus')(x)
```

```
x = MaxPooling1D(2)(x)
```

```
x = BatchNormalization()(x)
```

```
x = Dropout(.5)(x)
```

```
x = Flatten()(x)
```

```
x = Dense(50, activation = 'softplus', bias = True)(x)
```

```
x = Dense(50, activation = 'softplus', bias = True)(x)
```

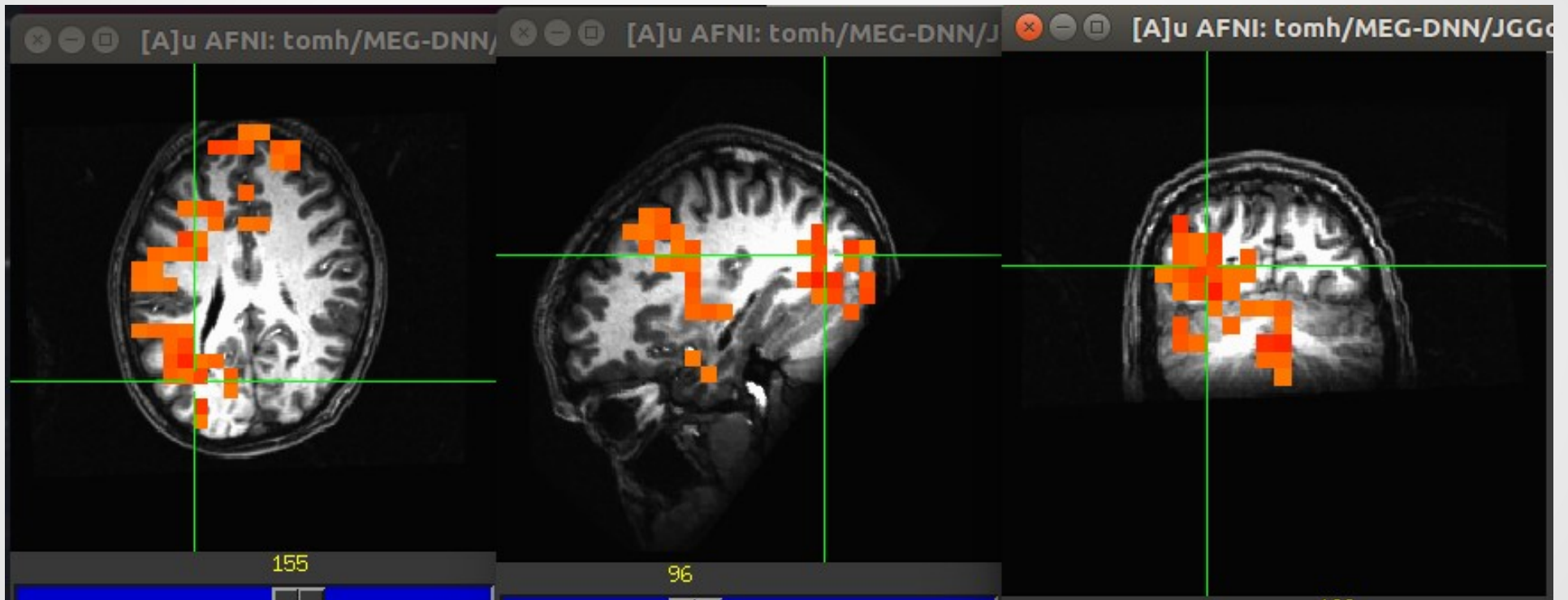
```
x = Dense(50, activation = 'softplus', bias = True)(x)
```

```
x = Dense(len(mlist), activation = 'softmax', bias = True)(x)
```

```
classifier = x
```

Predictive Ability

Each voxel's ability to classify the data alone (% over trials)




Beta band (15-30 Hz)

Loading Frozen Forward Solutions

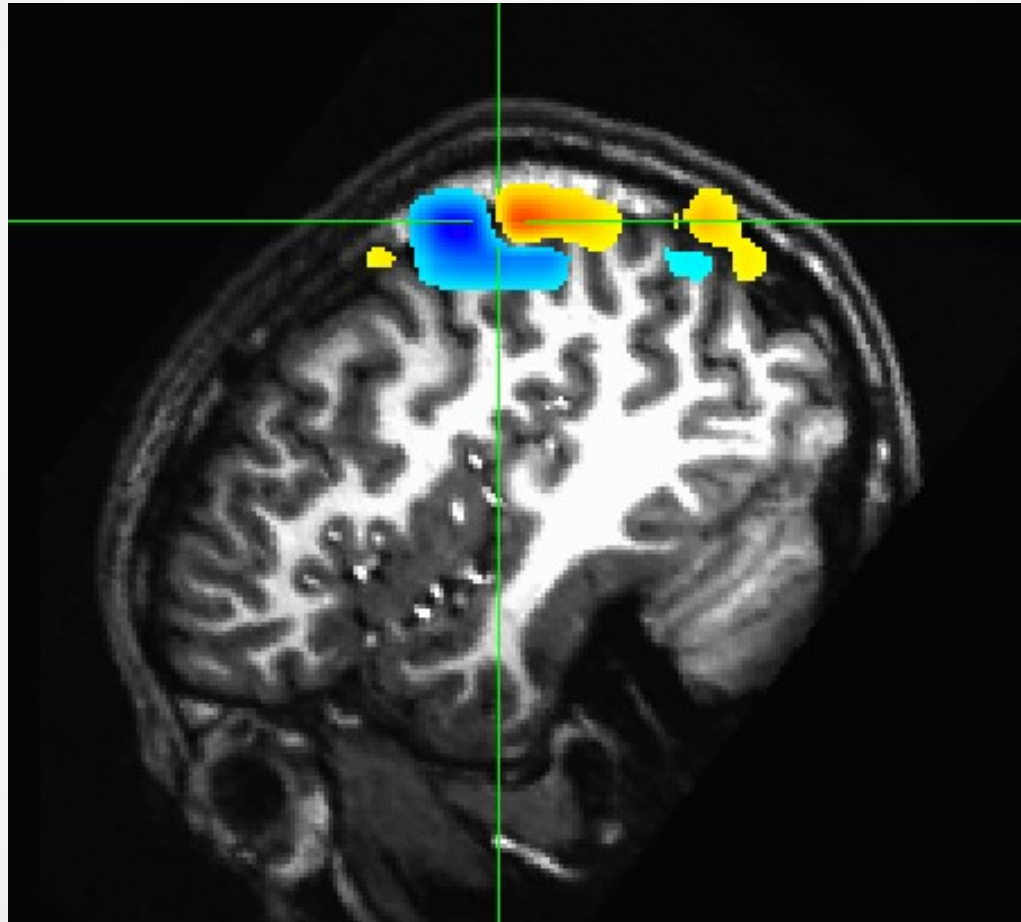
Keras/Tensorflow code that creates a frozen layer from forward solutions:

```
input_tensor = Input(shape = (M,))
x = Dense(30, activation = 'softplus')(input_tensor)
x = Dense(Nfwd, bias = False, activation = 'linear')(x)
Nfwd_tensor = x
fwd_layer = Dense(M, bias = False, activation = 'linear')
fwd_tensor = fwd_layer(Nfwd_tensor)
# Load forward solutions into the final layer's weights.
fwd_layer.set_weights([Dfwd]) # weights from sam_wts -B
fwd_layer.trainable = False

model = Model(input = input_tensor, output = fwd_tensor)
m1 = Model(input = input_tensor, output = Nfwd_tensor) # create after training model
m1.save("model.h5")
```



Near the Button Press



Nfwd_tensor prediction, given raw data point

BCI

- Use real-time beamformers to compute virtual channels in various parts of the brain
- Compute feature sets using the virtual RT data, for example, a low dimensional code layer of an autoencoder
- Train the DNN to use these features by conditioning the code layer's learning on the required output signal
- Time-dependent DNNs using LSTM and Transformers to decode neural language (* see me)

Thanks

Much of the real-time data-acquisition pipeline is still under development.

Participation is easy since most of the code is Python.

MEG Core Staff can help with customization to particular applications, and assistance with Deep Learning models.