The human brain contains 100 billion neurons

These neurons process information nonlinearly, thus making them difficult to study

Given the inputs and the outputs, how can we model the neuron’s computation?

Many models of increasing complexity have been developed.

The models I will be implementing are based on statistics.

- **Linear Models** — Linear Nonlinear Poisson (LNP) Model
  1. LNP using Spike Triggered Average (STA)
  2. LNP using Maximum Likelihood Estimates — Generalized Linear Model (GLM)
  3. Spike Triggered Covariance (STC)

- **Nonlinear Models**
  4. Generalized Quadratic Model (GQM)
  5. Nonlinear Input Model (NIM)
The Models - LNP

\[ r(t) = F(\bar{k} \cdot \bar{s}(t)) \]

**Knowns**
- \( \bar{s}(t) \) is the stimulus vector
- Spike times

**Unknowns**
- \( \bar{k} \) is a linear filter, defines the neuron's stimulus selectivity
- \( F \) is a nonlinear function
- \( r(t) \) is the instantaneous rate parameter of an non-homogenous Poisson process

The STA is the average stimulus preceding a spike in the output, where $N$ is the number of spikes and $s_{\text{spike}}(t)$ is the set of stimuli preceding a spike.

\[ STA = \frac{1}{N} \sum_{n=1}^{N} s_{\text{spike}}(t) \]


STA Implementation

- Filter length of 20 time steps
- Upsampling factor of 1
STA Implementation

- Resolution of the filter is determined by time interval between measurements

- We can artificially increase resolution by upsampling the stimulus vector:
  - Upsample by 1:
  - Upsample by 2:
Filter length of 15 time steps
For the STA, a common approach to finding the nonlinear response function is to use the histogram method.

Method creates bins for the generator signal, $\bar{k} \cdot \bar{s}(t)$, and plots average number of spikes for each bin.

$$r(t) = F(\bar{k} \cdot \bar{s}(t))$$
STA Implementation

- Filter length of 15 time steps
- Upsampling factor of 1
For filter validation, I
- created a stimulus with Gaussian random noise
- added an artificial filter at random points
- Recorded a spike for each instance of the artificial filter

If the STA code is working properly, and if none of the artificial filters overlap, then the code should exactly recover the artificial filter
STA Validation

Random Stimulus

No Spikes

Spikes After Filter

Random Stimulus with Filter

(time units of DTstim)
STA Validation

- **filter length**: 10
- **stimulus length**: 15000
- **20 spikes**: No overlap of filters in stimulus, STA code recovers exact filter
- **3000 spikes**: Substantial overlap of filters in stimulus, STA code recovers exact filter with some error
The Models - LNP

\[ r(t) = F(\bar{k} \cdot \bar{s}(t)) \]

- **Knowns**
  - \( \bar{s}(t) \) is the stimulus vector
  - Spike times

- **Unknowns**
  - \( \bar{k} \) is a linear filter, defines the neuron’s stimulus selectivity
  - \( F \) is a nonlinear function
  - \( r(t) \) is the instantaneous rate parameter of an non-homogenous Poisson process

Now we will approximate the linear filter using the Maximum Likelihood Estimate (MLE)

A likelihood function is the probability of an outcome $Y$ given a probability density function with parameter $\theta$

The LNP model uses the Poisson distribution

$$P(Y|\theta) = \prod_t \frac{(r(t)\Delta)^{y_t}}{y_t!} e^{-r(t)\Delta}$$

where $Y$ is the vector of spike counts binned at a resolution $\Delta$

We want to maximize a log-likelihood function

$$\mathcal{L} = \sum_{t=\text{spike}} log(r(t)) - \Delta \sum_t r(t) + \text{constants}$$

Can employ likelihood optimization methods to obtain maximum likelihood estimates for linear filter $\overline{k}$

- If we make some assumptions about the form of the nonlinearity $F$, the likelihood function has no non-global local maxima — gradient ascent!
  - $F(u)$ is convex in $u$
  - $\log(F(u))$ is concave in $u$

- I use $F(u) = \log(1 + \exp(u-c))$

$$\mathcal{L} = \sum_{t=\text{spike}} \log(\log(1 + \exp(\overline{k} \cdot \tilde{s}(t) - c))) - \Delta \sum_{t} \log(1 + \exp(\overline{k} \cdot \tilde{s}(t) - c))$$
GLM Implementation

- Originally coded a gradient descent method — took too many function evaluations
  - About 1000 iterations for a filter of length 15 at ~1s per function evaluation
- Next used a Newton-Raphson method — less iterations, but needed to compute Hessian
  - About 150 for a filter of length 15 at ~2s per function evaluation
- Need a quasi-Newton method
- Now use Matlab’s fminunc
  - About 10 – 150 iterations at ~1s per function evaluation
Filter length of 15 time steps
For the GLM, finding the parameters to the nonlinearity can be done at the same time as finding the filter.

Assume the parametric form $F(u)$ and include its parameter(s) in the optimization.

- Use $\log(1+\exp(x-c))$, fit offset $c$.
GLM Implementation

- Filter length of 15 time steps
- Upsampling factor of 1
We can also use regularization to add additional prior knowledge about solution attributes.

We know the filters should be smoothly varying in time:
- Penalize large curvatures in filter
- Laplacian gives us the second derivative; we want to maximize likelihood while minimizing the L2 norm of the Laplacian of the filter.

$$\mathcal{L} = \sum_{t=\text{spike}} \log(r(t)) - \Delta \sum_t r(t) - \lambda \|L^t\kappa\|_2^2$$

$\lambda$ is a parameter that is not explicitly part of the model, hence called a hyperparameter.
How to choose optimal $\lambda$?

For a variety of $\lambda$ values,
- fit model parameters using part of the data (80%)
- validate model on rest of the data (20%)
Filter length of 15 time steps

Upsampling factor of 2
GLM Implementation

- Filter length of 15 time steps
- Upsampling factor of 4
GLM Implementation

![Graph showing the effect of different values of lambda (\(\lambda\)) on time (ms). The graph compares \(\lambda = 0\) and \(\lambda = 360\).]
**Schedule**

- **PHASE I (October-December)**
  - Implement and validate the LNP model using the STA (October)
  - Develop code for gradient descent method and validate (October)
    - Done, but not efficient enough. I am currently using MATLAB’s fminunc command instead
  - Implement and validate the GLM with regularization (November-December)
  - Complete mid-year progress report and presentation (December)
Schedule

PHASE II (January-May)

- Implement quasi-Newton method for gradient descent (January)
- Implement and validate the LNP model using the STC (January-February)
- Implement and validate the GQM with regularization (February)
- Implement and validate the NIM with regularization using rectified linear upstream functions (March)
- Test all models (April)
- Complete final report and presentation (April-May)
References

# Figures