Real-time adaptive information-theoretic optimization of neurophysiology experiments

Presented by
Alex Roper

March 5, 2009

Goals

- How do neurons react to stimuli?
- What is a neuron's preferred stimulus?

Goals

- How do neurons react to stimuli?
- What is a neuron's preferred stimulus?
- Minimize number of trials.
- Speed - must run in real time.

Goals

- How do neurons react to stimuli?
- What is a neuron's preferred stimulus?
- Minimize number of trials.
- Speed - must run in real time.
- Emphasis on dimensional scalability (vision)

Challenges

- Typically high dimension
  - Model complexity - memory
  - Stimulus complexity - visual bitmap

Challenges

- Typically high dimension
  - Model complexity - memory
  - Stimulus complexity - visual bitmap
- Bayesian approach expensive
  - Estimation
  - Integration
  - Multivariate optimization

Challenges

- Typically high dimension
  - Model complexity - memory
  - Stimulus complexity - visual bitmap
- Bayesian approach expensive
  - Estimation
  - Integration
  - Multivariate optimization
- Limited firing capacity of a neuron (exhaustion)

Challenges

- Typically high dimension
  - Model complexity - memory
  - Stimulus complexity - visual bitmap
- Bayesian approach expensive
  - Estimation
  - Integration
  - Multivariate optimization
- Limited firing capacity of a neuron (exhaustion)
- Essential issues
  - Update a posteriori beliefs quickly given new data
  - Find optimal stimulus quickly

Neuron Model

$$p(r_t | \{x_t, x_{t-1}, ..., x_{t-t_k}\}, \{r_{t-1}, ..., r_{t-t_k}\})$$

Neuron Model

$$p(r_t | \{x_t, x_{t-1}, ..., x_{t-t_k}\}, \{r_{t-1}, ..., r_{t-t_k}\})$$

- The response r_t to stimulus x_t is dependent on x_t itself, as well as the history of stimuli and responses for a constant sliding window.

Neuron Model

$$p(r_t | \{x_t, x_{t-1}, ..., x_{t-t_k}\}, \{r_{t-1}, ..., r_{t-t_k}\})$$

- The response r_t to stimulus x_t is dependent on x_t itself, as well as the history of stimuli and responses for a constant sliding window.
- This is needed to measure exhaustion, depletion, etc.

Neuron Model

$$p(r_t | \{x_t, x_{t-1}, ..., x_{t-t_k}\}, \{r_{t-1}, ..., r_{t-t_k}\})$$

- The response r_t to stimulus x_t is dependent on x_t itself, as well as the history of stimuli and responses for a constant sliding window.
- This is needed to measure exhaustion, depletion, etc.

$$\lambda_t = E(r_t) = f\left(\sum_i \sum_{l=1} t_k k_{i,t-l} + \sum_{j=1}^{t_a} a_j r_{t-j},\right)$$

Neuron Model

$$p(r_t | \{x_t, x_{t-1}, ..., x_{t-t_k}\}, \{r_{t-1}, ..., r_{t-t_k}\})$$

- The response r_t to stimulus x_t is dependent on x_t itself, as well as the history of stimuli and responses for a constant sliding window.
- This is needed to measure exhaustion, depletion, etc.

$$\lambda_t = E(r_t) = f\left(\sum_i \sum_{l=1} t_k k_{i,t-l} + \sum_{j=1}^{t_a} a_j r_{t-j},\right)$$

- Filter coefficients $k_{i,t-l}$ represent dependence on the input itself.

Neuron Model

$$p(r_t | \{x_t, x_{t-1}, ..., x_{t-t_k}\}, \{r_{t-1}, ..., r_{t-t_k}\})$$

- The response r_t to stimulus x_t is dependent on x_t itself, as well as the history of stimuli and responses for a constant sliding window.
- This is needed to measure exhaustion, depletion, etc.

$$\lambda_t = E(r_t) = f\left(\sum_i \sum_{l=1} t_k k_{i,t-l} + \sum_{j=1}^{t_a} a_j r_{t-j},\right)$$

- Filter coefficients $k_{i,t-l}$ represent dependence on the input itself.
- a_j models dependence on observed recent activity.

Neuron Model

$$p(r_t | \{x_t, x_{t-1}, ..., x_{t-t_k}\}, \{r_{t-1}, ..., r_{t-t_k}\})$$

- The response r_t to stimulus x_t is dependent on x_t itself, as well as the history of stimuli and responses for a constant sliding window.
- This is needed to measure exhaustion, depletion, etc.

$$\lambda_t = E(r_t) = f\left(\sum_i \sum_{l=1} t_k k_{i,t-l} + \sum_{j=1}^{t_a} a_j r_{t-j},\right)$$

- Filter coefficients $k_{i,t-l}$ represent dependence on the input itself.
- a_j models dependence on observed recent activity.
- We summarize all unknown parameters as θ. This is what we're trying to learn.

Generalized Linear Models

- Distribution function (multivariate gaussian).
- Linear predictor, θ.
- Link function (exponential).

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)
- Laplace approximation to construct a Gaussian approximation of the posterior.

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)
- Laplace approximation to construct a Gaussian approximation of the posterior.
  - Set μ_t to the peak of the posterior.
  - Set covariance matrix C_t to negative inverse of Hessian of log posterior at μ_t.

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)
- Laplace approximation to construct a Gaussian approximation of the posterior.
  - Set μ_t to the peak of the posterior.
  - Set covariance matrix C_t to negative inverse of Hessian of log posterior at μ_t.
  - Compute directly?

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)
- Laplace approximation to construct a Gaussian approximation of the posterior.
  - Set μ_t to the peak of the posterior.
  - Set covariance matrix C_t to negative inverse of Hessian of log posterior at μ_t.
  - Compute directly?
  - Complexity is $O(td^2 + d^3)$

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)
- Laplace approximation to construct a Gaussian approximation of the posterior.
  - Set μ_t to the peak of the posterior.
  - Set covariance matrix C_t to negative inverse of Hessian of log posterior at μ_t.
  - Compute directly?
  - Complexity is $O(td^2 + d^3)$
    - $O(td^2)$ for product of t likelihood terms.
    - $O(d^3)$ for inverting the Hessian
  - Approximate $p(\theta_{t-1}|x_{t-1}, r_{t-1})$ as Gaussian

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)
- Laplace approximation to construct a Gaussian approximation of the posterior.
  - Set μ_t to the peak of the posterior.
  - Set covariance matrix C_t to negative inverse of Hessian of log posterior at μ_t.
  - Compute directly?
  - Complexity is $O(td^2 + d^3)$
    - $O(td^2)$ for product of t likelihood terms.
    - $O(d^3)$ for inverting the Hessian
  - Approximate $p(\theta_{t-1}|x_{t-1}, r_{t-1})$ as Gaussian
  - Now we can use Bayes' rule to find the posterior in one dimension.

Updating the Posterior

- Ideally, this runs in real time.
- Approximate the posterior as Gaussian
  - The posterior is the product of two smooth, log-concave terms.
  - (The GLM likelihood function and the Gaussian prior)
- Laplace approximation to construct a Gaussian approximation of the posterior.
  - Set μ_t to the peak of the posterior.
  - Set covariance matrix C_t to negative inverse of Hessian of log posterior at μ_t.
  - Compute directly?
  - Complexity is $O(td^2 + d^3)$
    - $O(td^2)$ for product of t likelihood terms.
    - $O(d^3)$ for inverting the Hessian
  - Approximate $p(\theta_{t-1}|x_{t-1}, r_{t-1})$ as Gaussian
  - Now we can use Bayes' rule to find the posterior in one dimension. $O(d^2)$.

Deriving the optimal stimulus

- Main idea: maximize conditional mutual information:

Deriving the optimal stimulus

- Main idea: maximize conditional mutual information:
- $I(\theta; r_{t+1} | x_{t+1}, x_t, r_t) = \mathcal{H}(\theta | x_t, r_t) - \mathcal{H}(\theta | x_{t+1}, r_{t+1})$.

Deriving the optimal stimulus

- Main idea: maximize conditional mutual information:
- $I(\theta; r_{t+1} | x_{t+1}, x_t, r_t) = \mathcal{H}(\theta | x_t, r_t) - \mathcal{H}(\theta | x_{t+1}, r_{t+1})$.
- This ends up being equivalent to minimizing the conditional entropy $\mathcal{H}(\theta | x_{t+1}, r_{t+1})$.

Deriving the optimal stimulus

- Main idea: maximize conditional mutual information:
- $I(\theta; r_{t+1}|x_{t+1}, x_t, r_t) = \mathcal{H}(\theta|x_t, r_t) - \mathcal{H}(\theta|x_{t+1}, r_{t+1})$.
- This ends up being equivalent to minimizing the conditional entropy $\mathcal{H}(\theta|x_{t+1}, r_{t+1})$.
- End up with equation for covariance in terms of Fisher information, J_{obs}.

Deriving the optimal stimulus

- Main idea: maximize conditional mutual information:
- $I(\theta; r_{t+1}|x_{t+1}, x_t, r_t) = \mathcal{H}(\theta|x_t, r_t) - \mathcal{H}(\theta|x_{t+1}, r_{t+1})$.
- This ends up being equivalent to minimizing the conditional entropy $\mathcal{H}(\theta|x_{t+1}, r_{t+1})$.
- End up with equation for covariance in terms of Fisher information, J_{obs}.
- We are able to solve for optimal stimulus using the Lagrange method for constrained optimization

Deriving the optimal stimulus

- Main idea: maximize conditional mutual information:
- $I(\theta; r_{t+1}|x_{t+1}, x_t, r_t) = \mathcal{H}(\theta|x_t, r_t) - \mathcal{H}(\theta|x_{t+1}, r_{t+1})$.
- This ends up being equivalent to minimizing the conditional entropy $\mathcal{H}(\theta|x_{t+1}, r_{t+1})$.
- End up with equation for covariance in terms of Fisher information, J_{obs}.
- We are able to solve for optimal stimulus using the Lagrange method for constrained optimization
- Thus, we have a system of equations in the Lagrange multiplier, and we can simply line search over it.

Deriving the optimal stimulus

- Complexity?

Deriving the optimal stimulus

- Complexity?
  - Rank-one matrix update and line search to compute μ_t and C_t.

Deriving the optimal stimulus

- Complexity?
  - Rank-one matrix update and line search to compute μ_t and C_t.$O(d^2)$.
  - Eigendecomposition of C_t.

Deriving the optimal stimulus

- Complexity?
  - Rank-one matrix update and line search to compute μ_t and $C_t . O(d^2)$.
  - Eigendecomposition of C_t. $O(d^3)$
  - Line search over Lagrange multiplier to compute optimal stimulus. $O(d^2)$
- $O(d^3)$ for the eigendecomposition isn't great...

Deriving the optimal stimulus

- Complexity?
  - Rank-one matrix update and line search to compute μ_t and $C_t . O(d^2)$.
  - Eigendecomposition of C_t. $O(d^3)$
  - Line search over Lagrange multiplier to compute optimal stimulus. $O(d^2)$
- $O(d^3)$ for the eigendecomposition isn't great...
- ...but because of our Gaussian approximation of θ, we can obtain C_t from C_{t-1} with a rank-one modification...

Deriving the optimal stimulus

- Complexity?
  - Rank-one matrix update and line search to compute μ_t and C_t.$O(d^2)$.
  - Eigendecomposition of C_t. $O(d^3)$
  - Line search over Lagrange multiplier to compute optimal stimulus. $O(d^2)$
- $O(d^3)$ for the eigendecomposition isn't great...
- ...but because of our Gaussian approximation of θ, we can obtain C_t from C_{t-1} with a rank-one modification...
- ...and there are eigendecomposition algorithms that can take advantage of this.

Deriving the optimal stimulus

- Complexity?
  - Rank-one matrix update and line search to compute μ_t and $C_t.O(d^2)$.
  - Eigendecomposition of C_t. $O(d^3)$
  - Line search over Lagrange multiplier to compute optimal stimulus. $O(d^2)$
- $O(d^3)$ for the eigendecomposition isn't great...
- ...but because of our Gaussian approximation of θ, we can obtain C_t from C_{t-1} with a rank-one modification...
- ...and there are eigendecomposition algorithms that can take advantage of this.
- This provides an average case runtime of $O(d^2)$ for the data considered, though the complexity is still $O(d^3)$ in the worst case.

What if θ is dynamic?

- Spike history terms

What if θ is dynamic?

- Spike history terms
  - Adds a linear term to a quadratic minimization problem for maximizing entropy.

What if θ is dynamic?

- Spike history terms
  - Adds a linear term to a quadratic minimization problem for maximizing entropy.
- Systematic trends in θ.

What if θ is dynamic?

- Spike history terms
  - Adds a linear term to a quadratic minimization problem for maximizing entropy.
- Systematic trends in θ.
  - Just add a random variable $N(0, C_t + Q)$ for known Q.

What if θ is dynamic?

- Spike history terms
  - Adds a linear term to a quadratic minimization problem for maximizing entropy.
- Systematic trends in θ.
  - Just add a random variable $N(0, C_t + Q)$ for known Q.
  - $\theta_{t+1} = \theta_t + \omega_t$.

What if θ is dynamic?

- Spike history terms
  - Adds a linear term to a quadratic minimization problem for maximizing entropy.
- Systematic trends in θ.
  - Just add a random variable $N(0, C_t + Q)$ for known Q.
  - $\theta_{t+1} = \theta_t + \omega_t$.

Results

- Simple, memoryless, visual cell
  - 25x33 bitmaps.
  - Results on average much better, and never worse, than random.

Results

- Simple, memoryless, visual cell
  - 25x33 bitmaps.
  - Results on average much better, and never worse, than random.
- Memoryful neuron (simple sine wave)
  - Outperformed random sampling for estimating spike history and stimulus coefficients.

Results

- Simple, memoryless, visual cell
  - 25x33 bitmaps.
  - Results on average much better, and never worse, than random.
- Memoryful neuron (simple sine wave)
  - Outperformed random sampling for estimating spike history and stimulus coefficients.
- Non-systematic time drift
  - Analogous to eye fatigue/exhaustion.
  - Outperformed random sampling for estimating spike history and stimulus coefficients.

Conclusion

- Approximations based on GLMs allow dramatically faster algorithm.

Conclusion

- Approximations based on GLMs allow dramatically faster algorithm.
- At worst, $O(n^3)$. on average, $O(n^2)$.

Conclusion

- Approximations based on GLMs allow dramatically faster algorithm.
- At worst, $O(n^3)$. on average, $O(n^2)$.
- Fast enough to run in real time even for high-dimensional problems.