Reverse correlation: linearizing black box functions

Reverse correlation has long been used to characterize biological neurons. The idea is to open the black box that is a biological neuron by using random stimuli to probe how the neuron responds. We present a series of random stimuli to a sensory neuron. By averaging those that generated a response, the obtain the spike-triggered average. The spike triggered average is equal to the weights of a linear filter that approximates the response of a neuron. Under this linearity assumption, it is also a neuron's preferred stimulus.

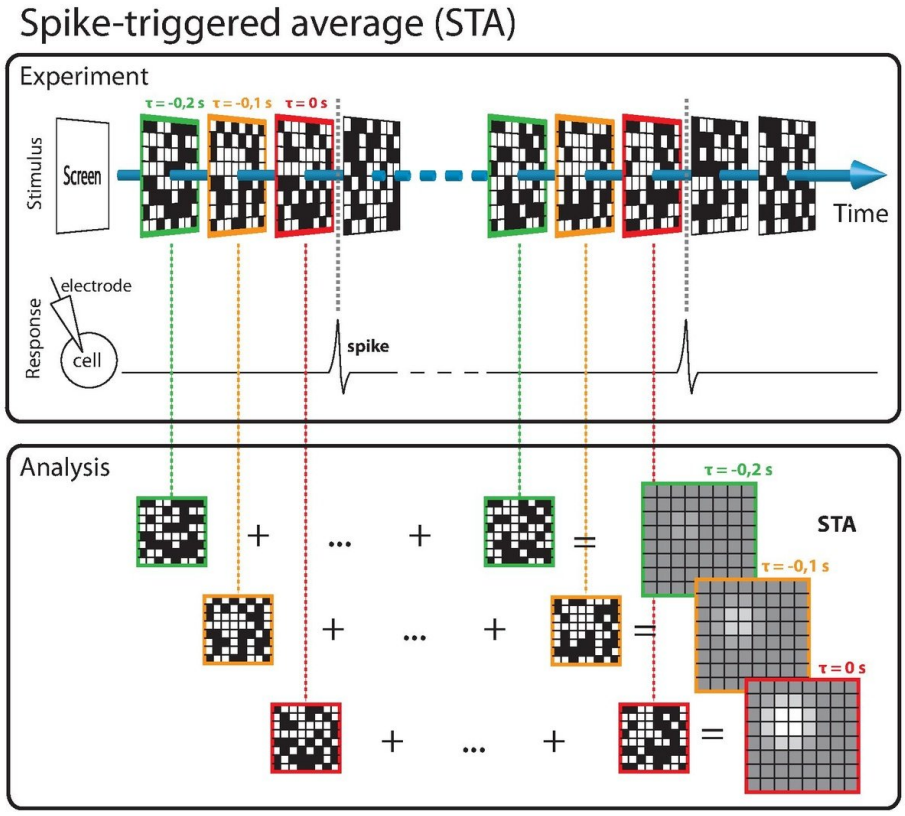


Figure 1: Reverse correlation in a nutshell. Source: https://en.wikipedia.org/wiki/File:Illustration_diagram_for_the_Spike-triggered_average.pdf

Reverse correlation has fallen out of favor as people have graduated to more complex models (e.g. GLMs and deep nets) with more complex stimuli (natural scenes) to probe brain representations. However, the machinery underlying reverse correlation is making somewhat of an unexpected comeback in different corners of machine learning. This is something that I'm sure will surprise many

machine learning people as well as neuroscientists. Digging deeper into these subfields yields some insights on improvements in reverse correlation that we can take back to computational neuroscience. In this article, I discuss two of these ideas:

- evolution strategies, which are black-box optimization methods
- locally interpretable models, which are linearization models to open up black box models like deep neural nets

Before I dig into these methods, let's dig into the theory of behind reverse correlation so we can know its deep structure.

The theory behind reverse correlation

Reverse correlation seemed pretty magical when I first encountered it! Put noise into a system, measure responses - out comes the kernel (?!). There's a good exposition of the ideas behind reverse correlation in Franz & Schölkopf (2006). Let's say that we are trying to characterize a nonlinear system that receives N inputs and outputs a noisy scalar y:

$$\mathbb{E}[y] = f(\mathbf{x}) \text{where } \mathbf{x} \in \mathbb{R}^N$$

Then, under certain regularity conditions, we can write the mean response function of the system $f(\mathbf{x})$ as a polynomial expansion, the Volterra expansion:

$$f(\mathbf{x}) \approx h_0 + (\mathbf{x} - \mathbf{x}_0)^T \cdot h_1 + (\mathbf{x} - \mathbf{x}_0)^T h_2 (\mathbf{x} - \mathbf{x}_0) + \dots$$

Note that the kernels h_i have increasing order: h_0 is a scalar, h_1 a vector, h_2 a matrix, h_3 a three-index tensor, etc. The form of the expansion is similar to a Taylor expansion, and it's tempting to identify the two. However, the Volterra expansion is more general than a Taylor expansion. For example, the function $f(x) = \exp(-1/x^2)$ has 0 derivatives everywhere. Hence, its Taylor expansion around 0 is exactly 0. However, it has a perfectly good non-null Volterra expansion, which can be found through polynomial regression. Here's a notebook that shows you how it's done.

To estimate the coefficients of the Volterra expansion, conventional reverse correlation first rearranges the Volterra expansion into an equivalent series of functionals:

$$f(\mathbf{x}) = \sum_i G_i[\mathbf{x}]$$

And imposes that these functionals are orthogonal to each other:

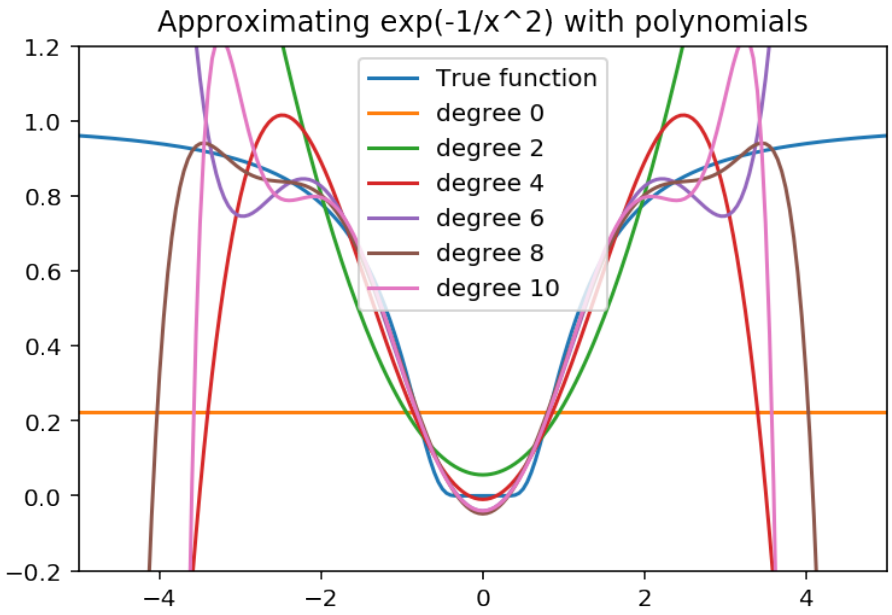


Figure 2: Approximating a function with zero derivatives with a Volterra expansion

$$\mathbb{E}_D[G_i(x)G_j(x)] = 0 \text{ for } i \neq j$$

Here D is the data distribution. Orthogonality is imposed via a Gram-Schmidt orthogonalization process:

1. Set G_0 to the output of the Volterra kernel of order 0
2. Set G_1 to the output of the Volterra kernel of order 1, and back-project G_0 from G_1 to make G_1 orthogonal to G_0
3. Set G_2 to the output of the Volterra kernel of order 2, and back-project G_0 and G_1 from G_2 to make G_2 orthogonal to both, etc.

It turns out that the expectations of every order are straightforward to calculate when the data distribution D is normal. The 0'th order and 1st order functionals are simply the constant and linear Volterra kernels. The 2nd order functional is the quadratic kernel minus a correction for the diagonal terms of the kernel, since the square of a normal distribution has a mean equal to the square of the variance, $\mathbb{E}[x^2] = \sigma^2$. Thus:

$$G_0 = k_0 G_1 = (x - x_0)^T k_1 G_2 = (x - x_0)^T k_2 (x - x_0) - \sigma^2 \sum_j k_2^{jj}$$

Since each functional are orthogonal over the data distribution, we can easily calculate them using an iterative procedure:

1. Estimate the Wiener kernel k_0 from the measured response y
2. Project the 0'th order functional out of y to obtain y_{res}
3. Estimate the Wiener kernel k_1 from the residual y_{res}, etc.

What does *estimate* mean in this context? Least-squares! To estimate the zero-order kernel, we have:

$$\hat{k}_0 = \arg\min_{k_0} \sum (y - k_0)^2 = \bar{y}$$

For the first-order kernel:

$$\hat{k}_1 = \arg\min_{k_1} \sum (y - \bar{y} - (x - x_0)^T k_1)^2 = ((x - x_0)^T (x - x_0))^{-1} (x - x_0)^T (y - \bar{y}) \approx \frac{1}{N\sigma^2} (x - x_0)^T (y - \bar{y})$$

Here the approximate sign comes from the fact that with normally distributed data, $(x - x_0)^T (x - x_0) \approx N\sigma^2 I$. The second order kernel is a little more tricky. We first define $r = y - \bar{y} - (x - x_0)^T$. Then:

$$k_2 = \arg\min_{k_2} \sum \left(r - (x - x_0)^T k_2 (x - x_0) + \sigma^2 \sum_j k_2^{jj} \right)^2 = \text{unstack}((D^T D)^{-1} D^T r)$$

The columns of the matrix D is composed of all the cross-products $x_i x_j - \delta_{ij}\sigma^2$ for $i \geq j$. The operator unstack takes this linear representation of the coefficients and transforms them to a matrix. Here again, we find that for normally distributed inputs, the non-diagonal terms in the covariance matrix $D^T D$ have an expected value of zero. The inverse of the diagonal is a bit more tricky to calculate, but a little bit of grinding shows that we pick up a scalar factor $\frac{1}{2!\sigma^4}$. In fact, we can show that the estimate of the kernels can be written as:

$$k_i(j_1, j_2, \ldots j_i) = \frac{1}{i!\sigma^{2i} N} [y - \sum_{m=0}^{i-1} G_m(\mathbf{x})] x_{j_1} x_{j_2} \ldots x_{j_i}$$

Where G_j are the orthogonalized Wiener functionals (equation 2, Korenberg et al. 1988). Notice the $i!$ factors, which look an awful lot like what you'd find in a Taylor expansion. This formula actually works! Here's a notebook to show you how it's done for a second order Wiener expansion.

Is your head spinning yet? Mine sure was! Grinding through the kernels one by one makes a few things clear:

1. Wiener-Volterra analysis isn't magic. It's orthogonal polynomial regression estimated through least-squares, with approximations that are enabled by the use of a very special distribution of the inputs, which means that expectations of different moments can be analytically derived.
2. The Wiener kernel definitions change when you change the distribution of the input. This is because orthogonalization is dependent on the input distribution.
3. The easy estimation of the kernels via backprojection and simple multiplication with cross-products of the inputs is lost when the input distribution is not normal.

Casting Wiener-Volterra analysis as polynomial regression fit via least-squares immediately opens up many extensions:

1. Finding better estimates via the minimum variance unbiased estimator, that is, maximum likelihood
2. Finding better estimates by better modeling the noise, for example, taking into account the Poisson nature of neural noise
3. Generalizing to non-normally distributed ensembles of inputs, again using maximum likelihood
4. Using alternative expansions, for example, cascades of linear-nonlinear filters, fit with stochastic gradient descent

Remarkably, some of the ideas behind Wiener-Volterra theory - local polynomial expansion, probing points randomly - have made their way into far-flung machine learning methods. Let's take a look.

Evolution strategies (ES)

Evolution strategies (ES) are a family of black-box, gradient-free optimizers for potentially non-differentiable functions. The most basic component of an optimizer is the computation of the gradient. If the gradient is unaccessible because of discontinuities, or because it is very expensive to calculate, we can obtain a *smoothed* gradient via NES.

The setup is quite clever. We have a function f that we are trying to optimize. Instead of computing its derivative $\nabla f(z)$ directly, we compute the derivative of the expectation of the function, $\nabla \mathbb{E}_\pi[f(z)]$ in a local neighborhood $\pi(\theta, z)$. Think of this expectation as a smoothed derivative.

To compute the expected value, we expand it via its definition, and obtain:

$$\nabla_\theta \mathbb{E}_\pi(z,\theta)[f(z)] = \nabla \int \pi(z,\theta) f(z) dz = \int \nabla \pi(z,\theta) f(z) dz = \int \nabla \log \pi(z,\theta) \pi(z,\theta) f(z) dz$$

The last equality, which is also called the log-derivative trick is derived from:

$$\nabla \log \pi(z, \theta) = \frac{\nabla \pi(z, \theta)}{\pi(z, \theta)}$$

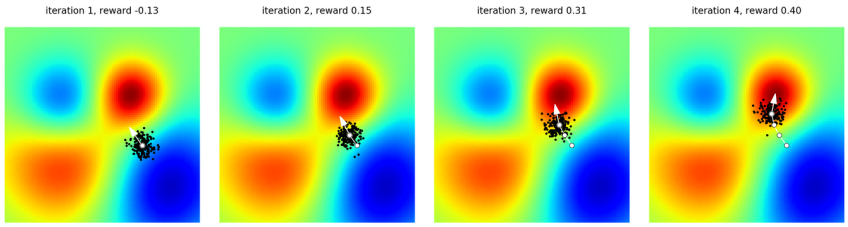
A standard neighborhood which is often used in practice is the isotropic gaussian centered at μ with a standard deviation σ. If we take a Monte Carlo estimate of the expectation, we find an approximation for the gradient as:

$$\nabla E[f] \approx \frac{1}{N} \sum_{1}^{N} \nabla_\mu \log N(\mu, \sigma^2) = \frac{1}{N} \sum_{1}^{N} \frac{1}{\sigma^2}(z - \mu) f(z)$$

Notice this remarkable equation: this is exactly the response-triggered average with an isotropic gaussian stimulus ensemble. Thus, we can view reverse correlation as estimating an approximate gradient of the response function of a neuron. This reveals an unexpected link between reverse correlation and optimization:

> The response-triggered average tells us how we should change a stimulus locally to increase the response of the system.

Taking ES to reverse correlation



Above: ES optimization process, in a setting with only two parameters and a reward function (red = high, blue = low). At each iteration we show the current parameter value (in white), a population of jittered samples (in black), and the estimated gradient (white arrow). We keep moving the parameters to the top of the arrow until we converge to a local optimum. You can reproduce this figure with this notebook.

Figure 3: From the OpenAI blog, https://openai.com/blog/evolution-strategies/

The unexpected link between reverse correlation and optimization opens up a lot of different avenues. For instance, we could adapt ES to maximize the response of a neuron in a closed-loop fashion. This strategy has the advantage of being trivial to compute and highly intuitive (i.e. no complaints from reviewer 2 about the method being inscrutable).

ES is also embarassingly parallel, which has been used to good effect in the context of reinforcement learning - OpenAI showcased ES with 1024 parallel

evaluations. We could consider evaluating the gradient of a neuron's response function in parallel with another neuron - perhaps in a different animal, in a different lab - and optimize a stimulus that jointly maximizes the sum of the responses. Indeed, we don't have to focus on maximizing just the mean response of a neuron: we can maximize diversity of responses, their dimensionality, etc. This works as long as the metric that we're measuring is a scalar function of the measured responses.

ES comes in a variety of different flavors that aim to increase the efficiency of the method. Two of the most popular variants are:

- CMA-ES (Covariance-matrix-adaptation). This takes the derivative of the density with respect to the covariance of the normal distribution, in addition to its mean. Thus, the covariance stimulus ensemble is tweaked so the method adapts to capture the largest dimensions of descent.
- NES (Natural evolution strategies). This uses natural gradients rather than gradients, which naturally deals with rescaling of the axes.

Both of these more efficient variants could be applied to find preferred stimuli in a population of neurons.

Finally, because the ES objective is a Monte Carlo estimate, we can import many of the tricks from the variance reduction literature to increase the efficiency of reverse correlation. For example, we can use control variates or use stratified sampling strategies. My favorite trick to increase the efficiency of estimation: pairing every stimulus with its negative. This means that the mean of the realized stimulus ensemble is exactly the mean of the distribution. This antithetic sampling can, in some circumstances, increase the efficiency of the estimation of the preferred stimulus.

Locally interpretable models

There's another area where the ideas underlying reverse correlation have recently come together: locally interpretable models. Suppose that we have a complex black box model that we are trying to better understand. This black box model can be, for example, a deep neural network. We can locally approximate this complex function with a simpler, local expansion. This local explanation does not capture all the subtleties of the model, but it can reveal insights that are useful for humans that want to understand the model.

In general, we try to find a model g which approximates a complex function f according to a weight window π:

$$\arg\min_{g \in G} L(f, g, \pi) + \Omega(g)$$

Here L is a loss function and $\Omega(g)$ captures the complexity of g. In LIME, or locally interpretable model explanations, the authors suggest to use a Gaussian-

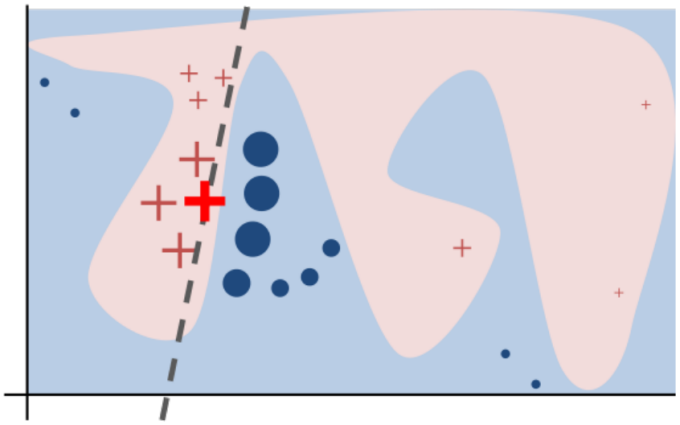


Figure 3: Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful.

Figure 4: Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets pre- dictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful. From Ribeiro et al. 2016

shaped neighborhood around an anchor a, and a sum-of-squares loss, leading to the weighted loss:

$$L = \sum_i \exp\left(-\frac{||x_i - a||^2}{\sigma^2}\right)(f(x_i) - g(x_i))^2$$

Several choices are possible for $\Omega(g)$. For example, we may privilege functions that focus on few input dimensions; we can impose that g is sparse in parameter space using an L1 penalty as $\Omega(g)$. The model class G can be anything that is "easy to interpret" for humans. This could mean:

- a GAM
- a decision tree
- a linear model

If we choose no penalty for complexity, and choose a linear model class, our model and error function is equivalent to reverse correlation and evolutionary strategies. This highlight something that's underappreciated about reverse correlation:

> The spike-triggered average is a simplified, locally valid explanation for how a neuron works. While complex models like deep nets can capture more of the variance in the data, the goal in reverse correlation is different: explaining neurons to humans.

Taking LIME to reverse correlation

An underappreciated fact about reverse correlation is that the size of the neighborhood around which we're linearizing is arbitrary and that different neighborhoods will return different responses. Many articles have documented that receptive fields appear to change with changing stimulus ensembles, for example natural vs. artificial stimuli. However, many of these articles have explained it in terms of some special response of neurons to natural image statistics; for instance, that they increase the degree of feedforward inhibition they receive. In fact, the same phenomenon could happen simply by tweaking the standard deviation of the noise with which we probe neurons.

In fact, each of these factors is arbitrary:

- the size of the neighborhood (big vs. small)
- the shape of the neighborhood (isotropic Gaussian vs. non-isotropic vs. natural stimuli)
- the center of the neighborhood (gray image or another reference)

Some of the most interesting applications of nonlinear systems identification have played with each of these factors. For example, by shifting the center of the neighborhood to an appropriate anchor point, we can locally linearize quadratic responses (i.e. complex cells) by choosing an appropriate anchor point. This is

the idea behind Movshon et al. (1978), where the line weighting function of a complex cell is found by using another line in the center as an anchor:

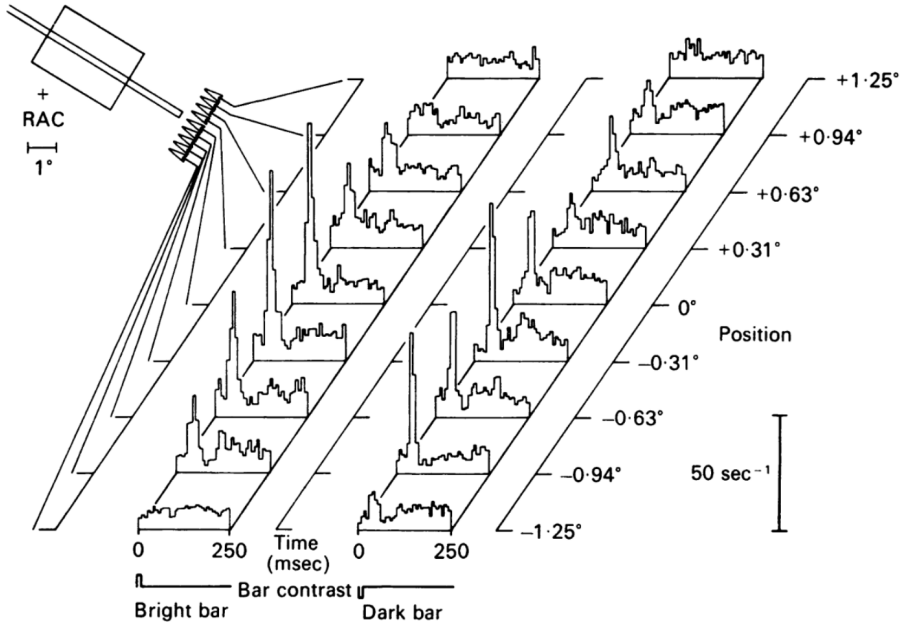


Fig. 5. An experiment to determine the line-weighting function of a complex cell. The minimum response field of the neurone is shown at the upper left, with the position of the right *area centralis* (RAC); a bar of the width used to obtain the line-weighting function is superimposed to scale on the receptive field; the bars in the experiment were all 10° long. The arrows indicate the nine bar positions tested; they were separated by the width of the bars (0·31°). Each stimulus was briefly flashed on for 32 msec every 250 msec, as indicated below each column of histograms.

Figure 5: Local linearization of receptive fields

There are some other excellent nuggets in the LIME paper that could serve us well. For instance, they suggest using a maximally diverse ensemble of examples (SP-LIME) to illustrate the functioning of the models for humans. A similar idea was used in Cadena et al. (2018) to visualize diverse features that triggered large responses in early stages of a deep neural net around an anchor point (invariant or equivariant features).

Finally, we can have our cake and eat it too: we can fit very complex models of neurons that explain much of the variance in sensory systems using deep nets, and visualize how these models work using locally linear explanations.

Conclusions

Both ES and LIME use a similar local linearization strategy as reverse correlation. From ES, we learn that the reverse correlation estimate gives us a local approximation for the gradient, which can be used as a starting point for closed-loop experiments. From LIME, we see that we can perform reverse correlation centered around different neighborhoods, yielding different results. By following these new research trends in machine learning, we can find new innovative ways of opening the black boxes that are biological neurons.