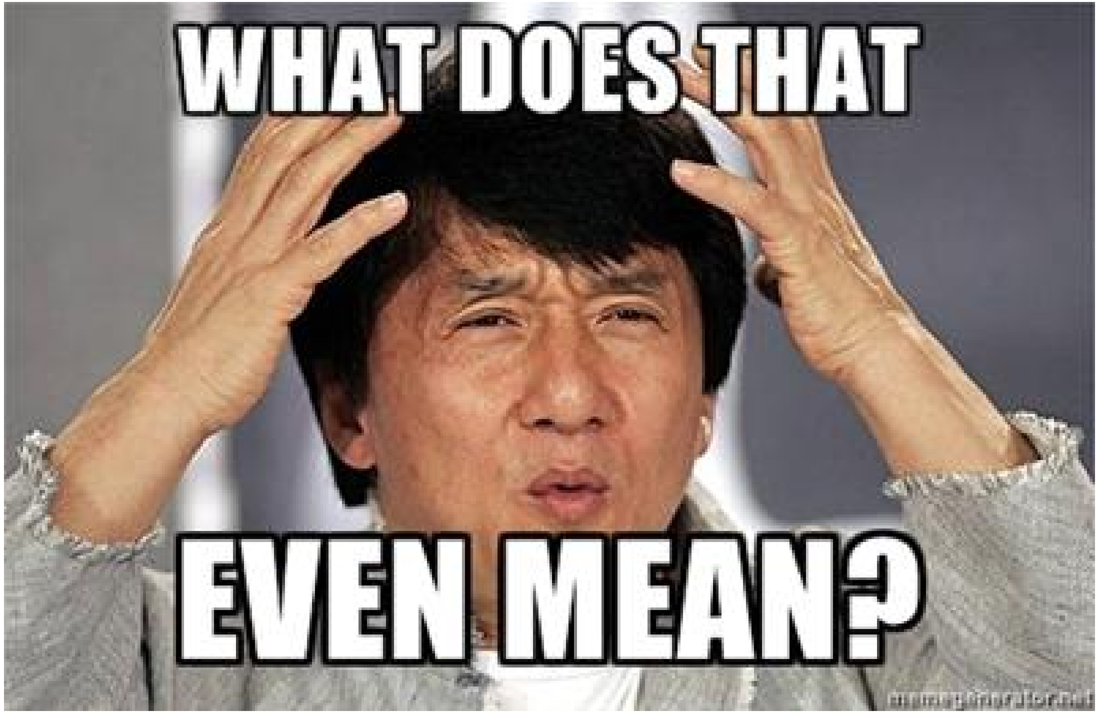
Gaussian Processes

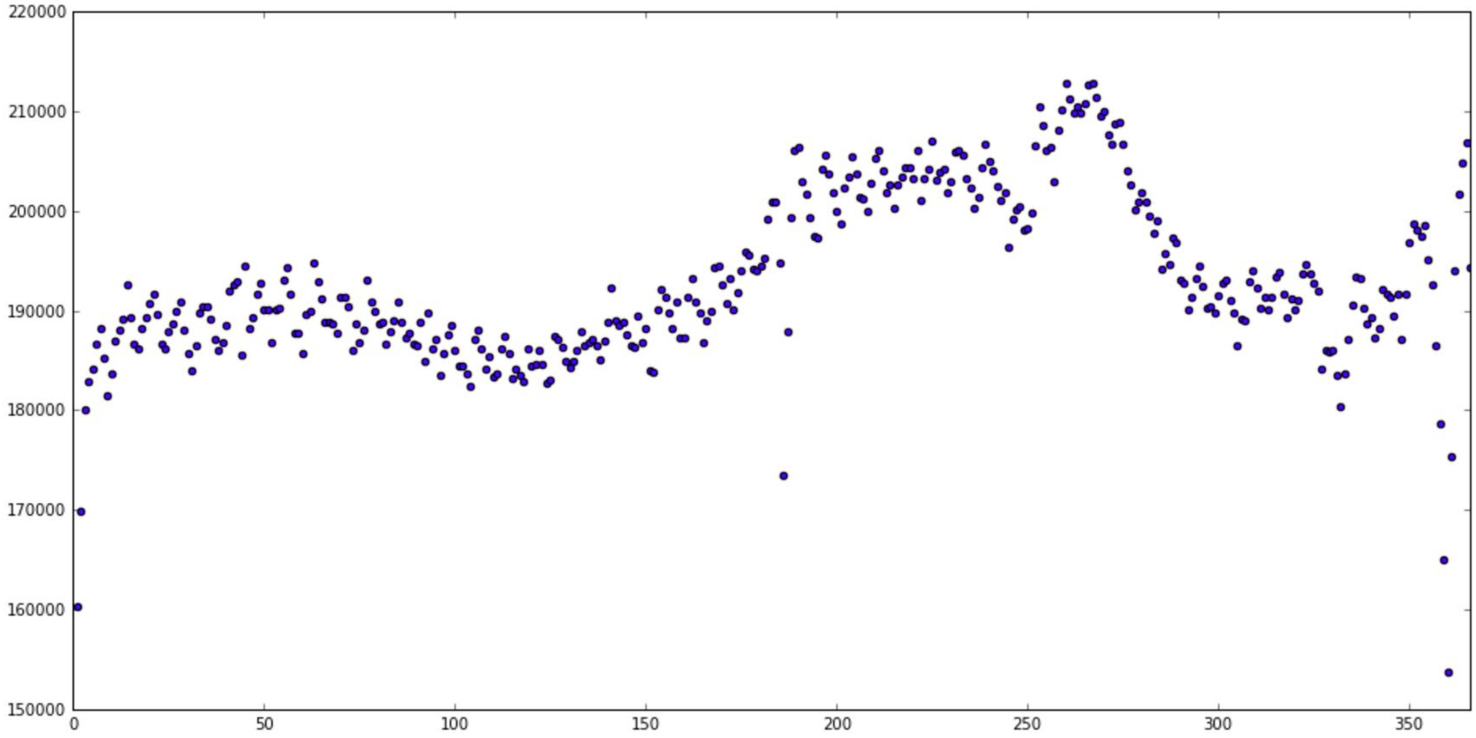
An Introduction

Gaussian Processes are the generalization of a Gaussian distribution over a finite vector space to a function space of infinite dimension

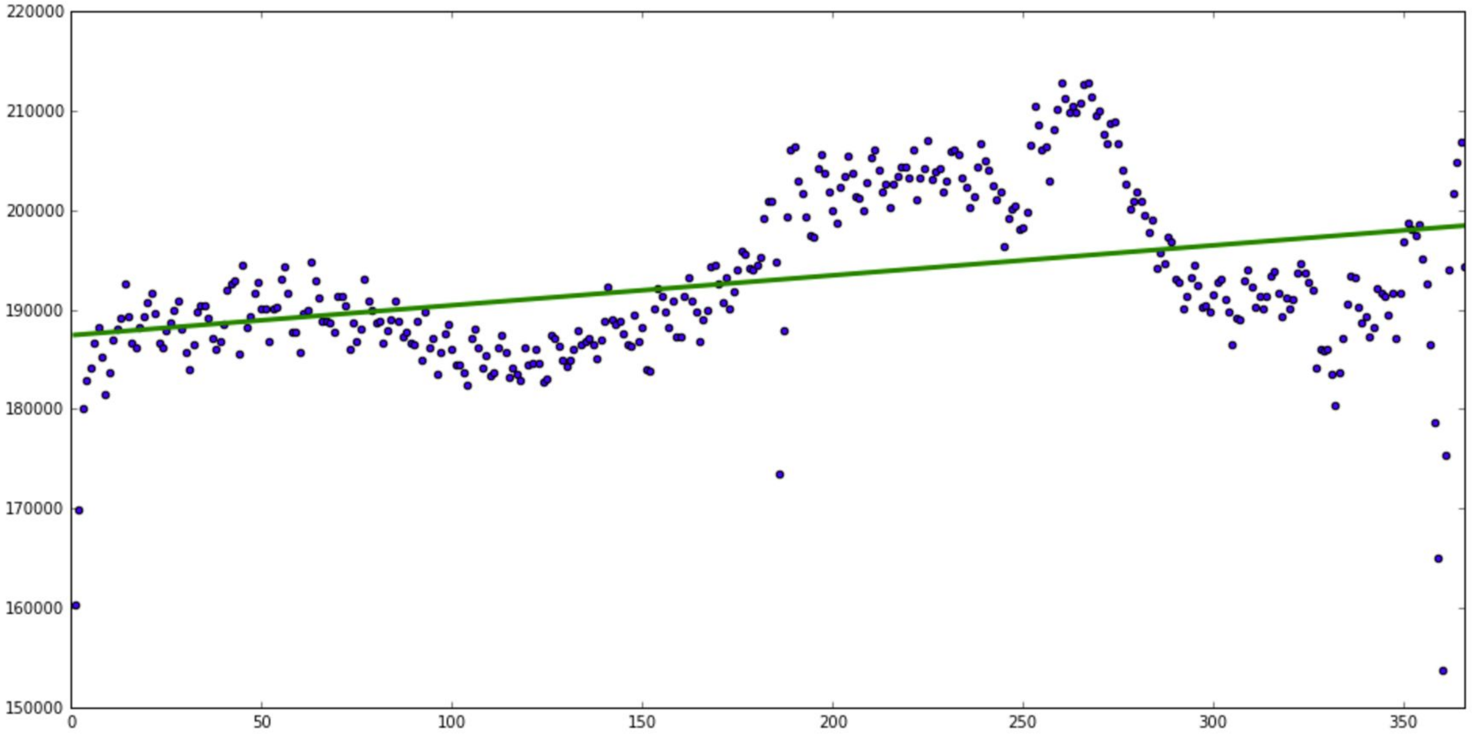

WHAT DOES THAT
EVEN MEAN?

Let’s start with Linear Regression :)

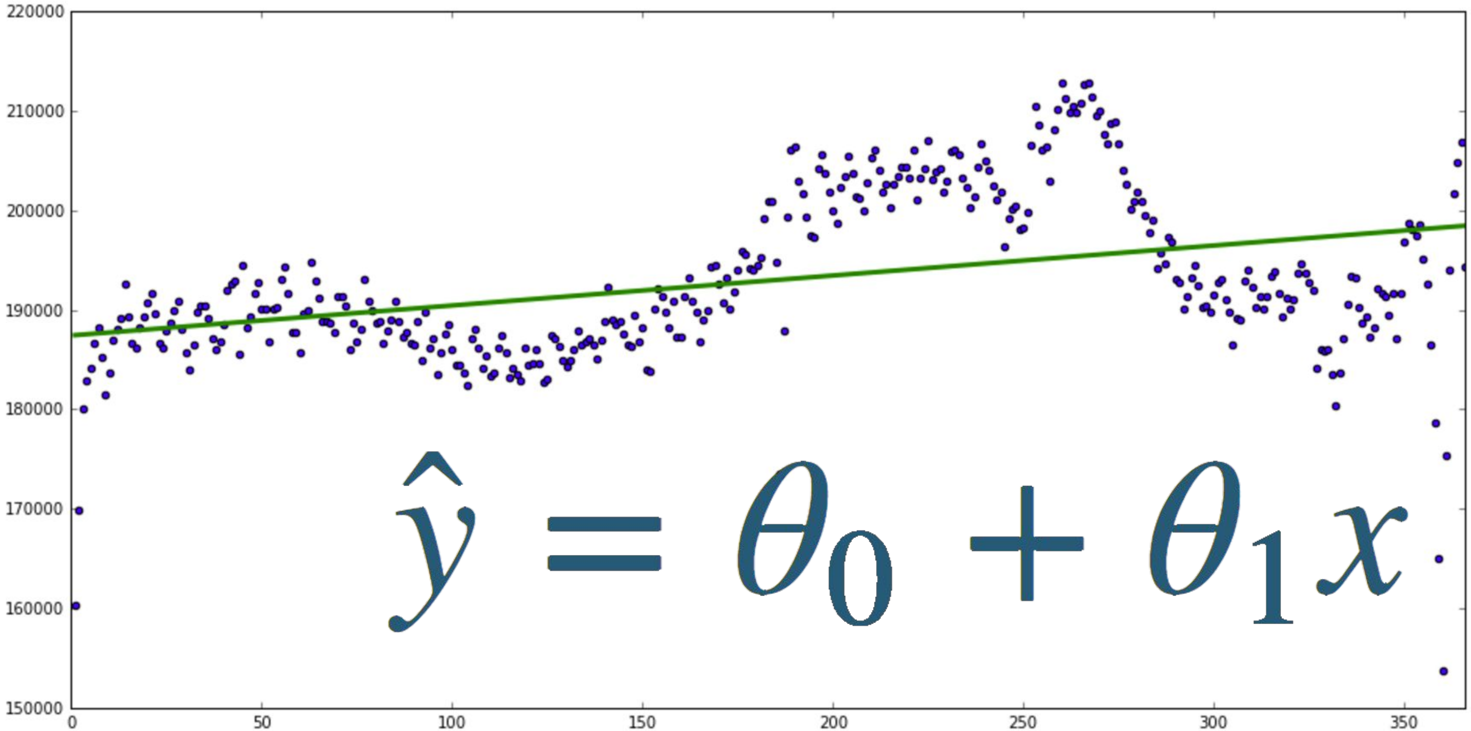
Birth frequencies by date



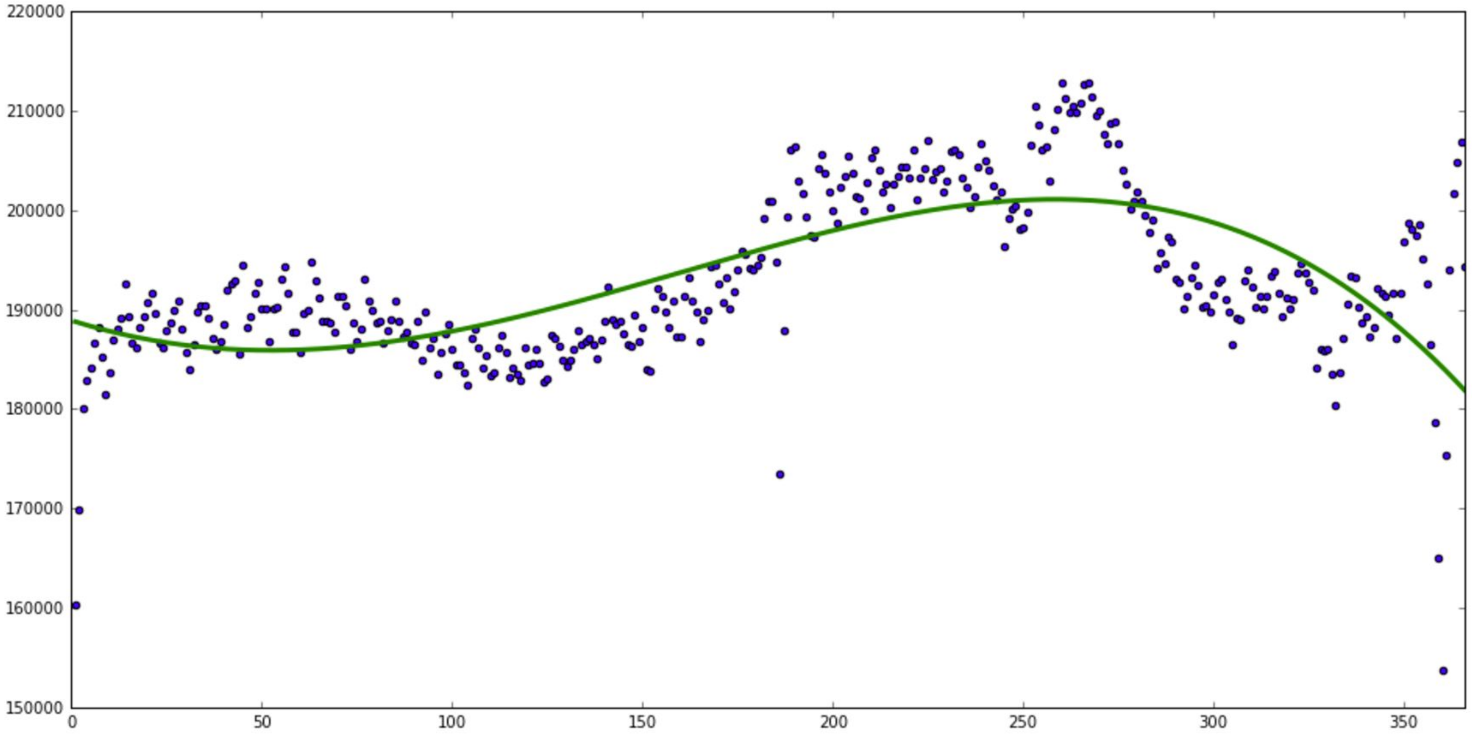
Birth frequencies by date



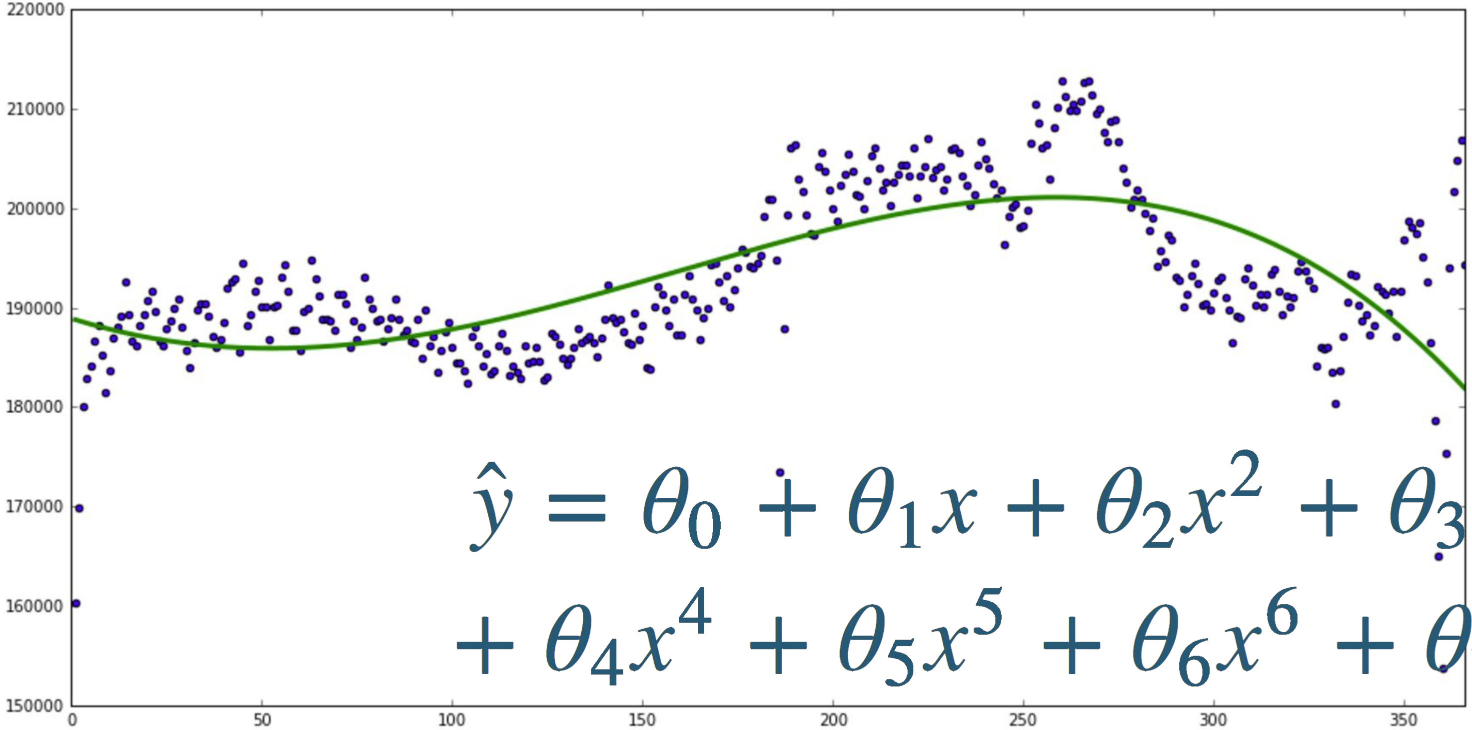
Birth frequencies by date



Birth frequencies by date

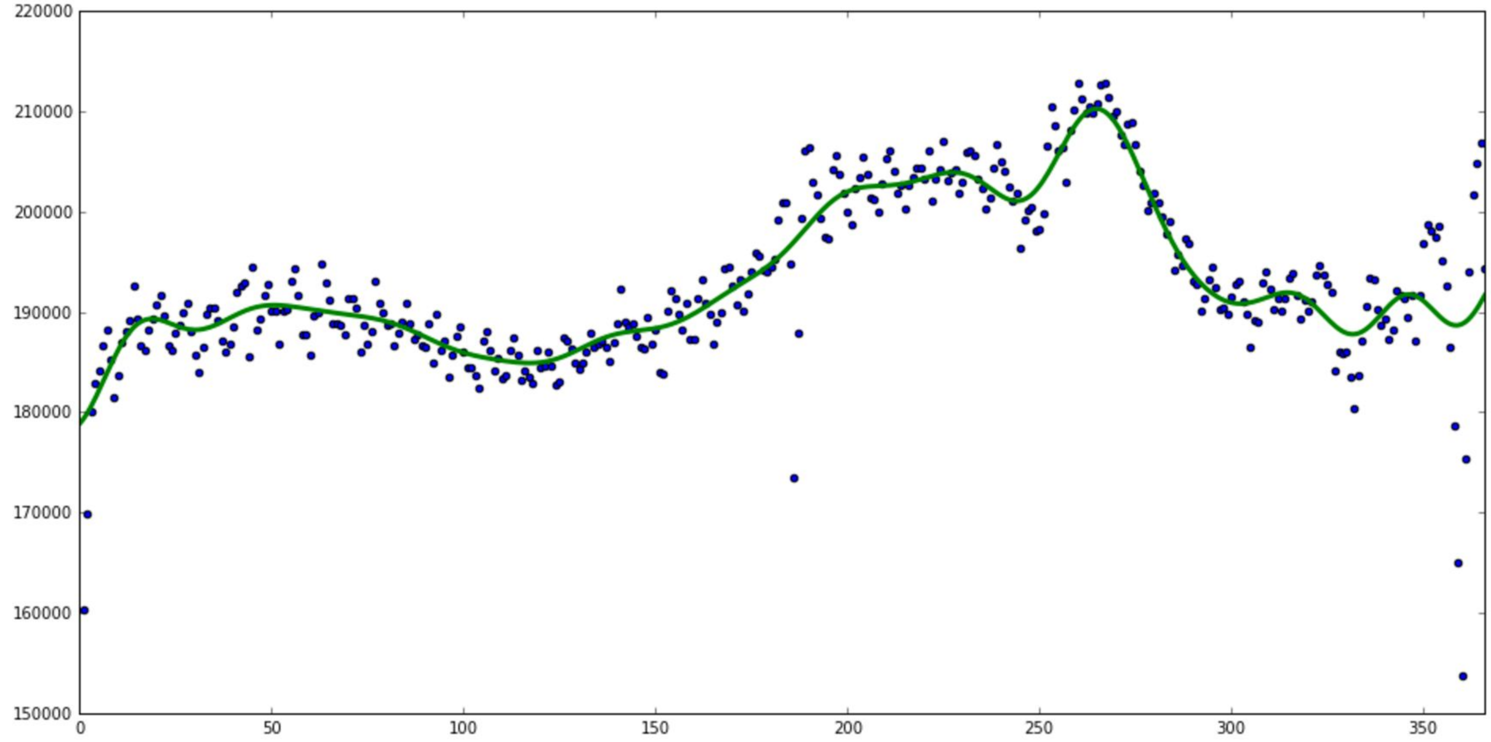


Birth frequencies by date



$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 + \theta_7 x^7$$

A better way?



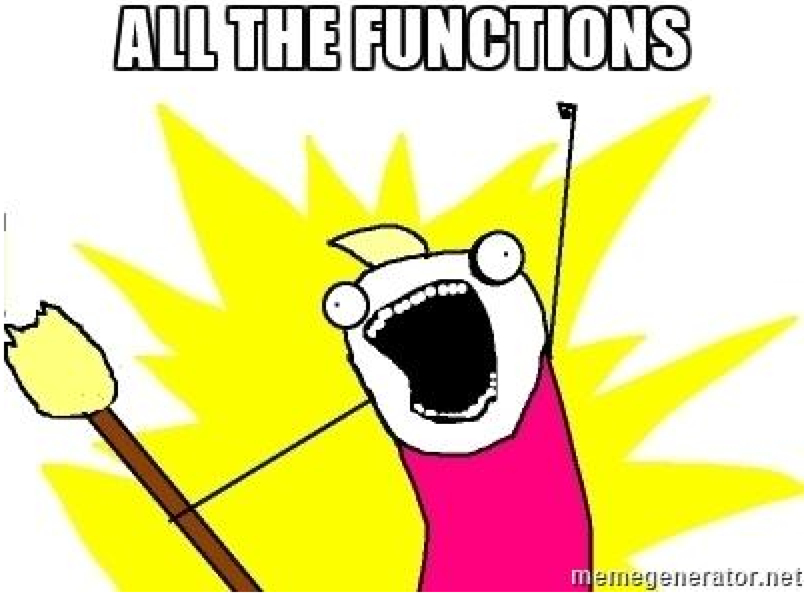
220000
210000
200000
190000
180000
170000
160000
150000
0
50
100
150
200
250
300
350

A better way?

$$y = f(x) + \epsilon$$

A better way?

Consider



...within reason :)

Gaussian Processes in a nutshell

Over some restricted input space...

- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x

Gaussian Processes in a nutshell

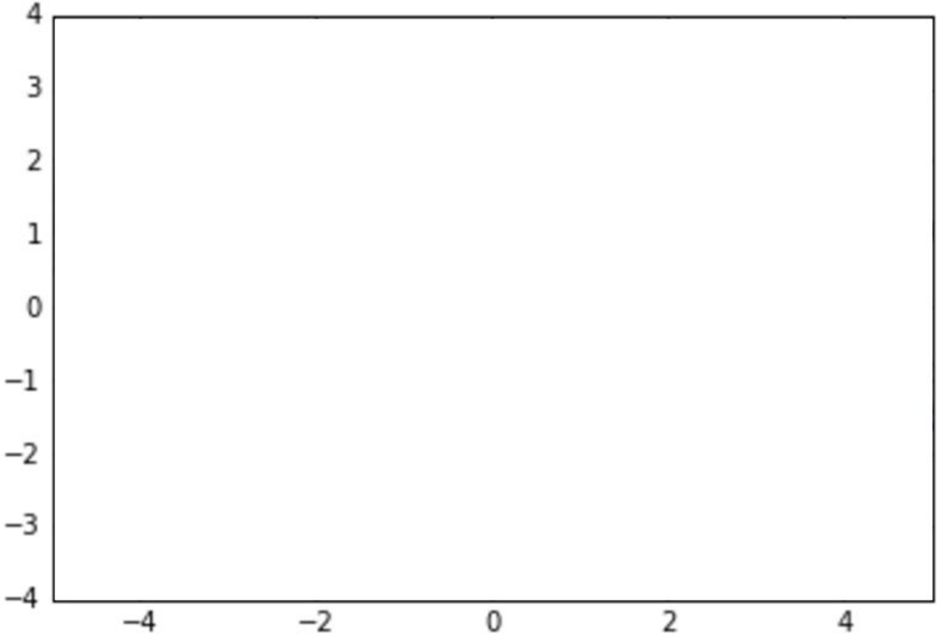
Over some restricted input space...

- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x

Gaussian Processes in a nutshell

Over some restricted input space...

- Come up v ... ns.
- Observe s
- Come up v ... ıctions.
- Sample frc ... predictions for unobse



Gaussian Processes in a nutshell

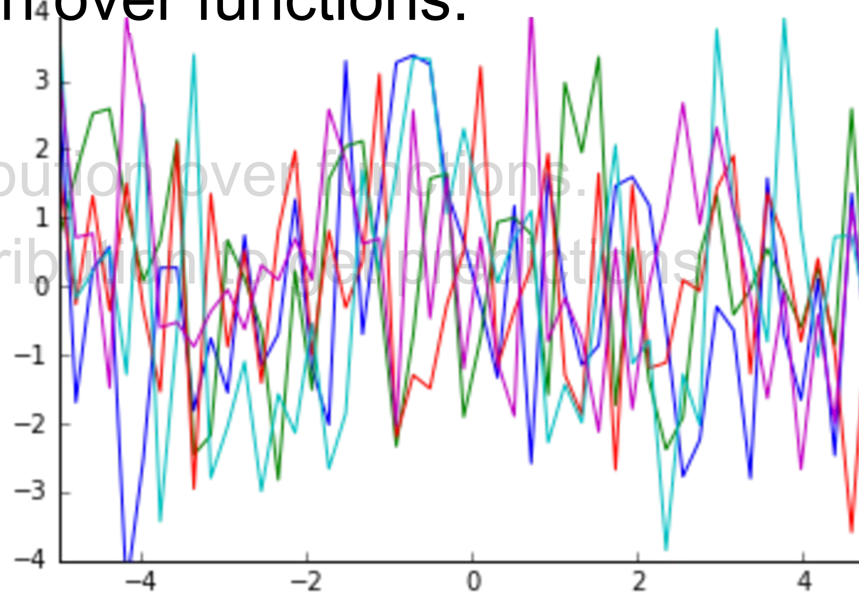
Over some restricted input space...

- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x

Gaussian Processes in a nutshell

Over some restricted input space...

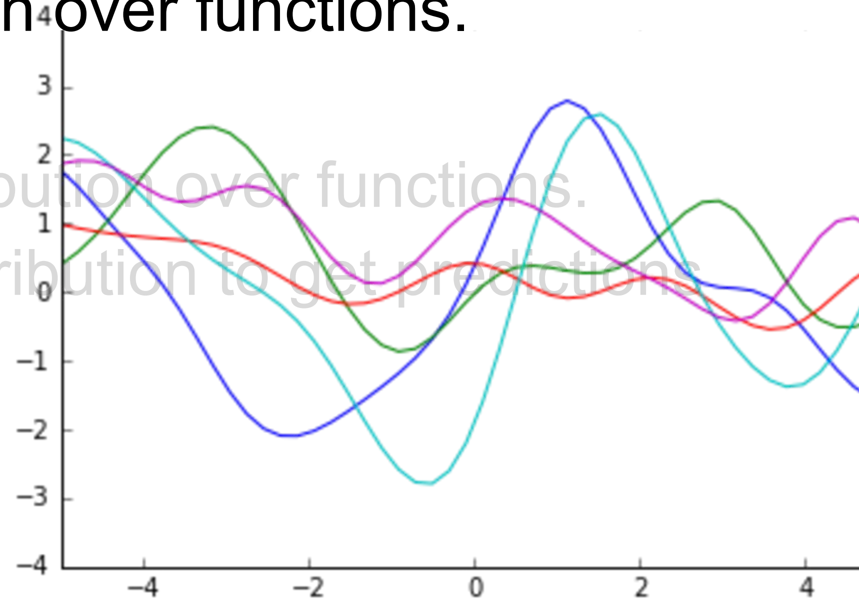
- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x



Gaussian Processes in a nutshell

Over some restricted input space...

- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x



Gaussian Processes in a nutshell

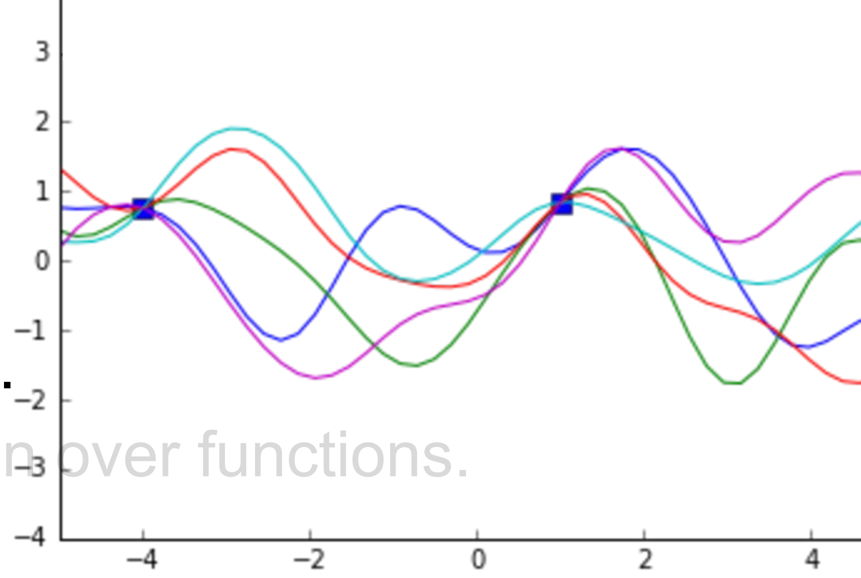
Over some restricted input space...

- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x

$$f(-4) = 0.757$$

$$f(1) = 0.841$$

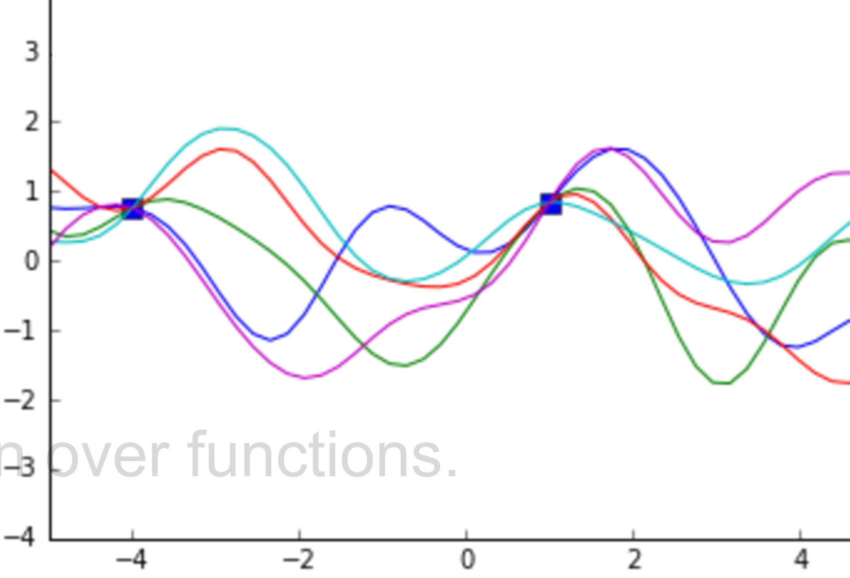
Gaussian Processes in a nutshell



Over some restricted input space...

- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x

Gaussian Processes in a nutshell

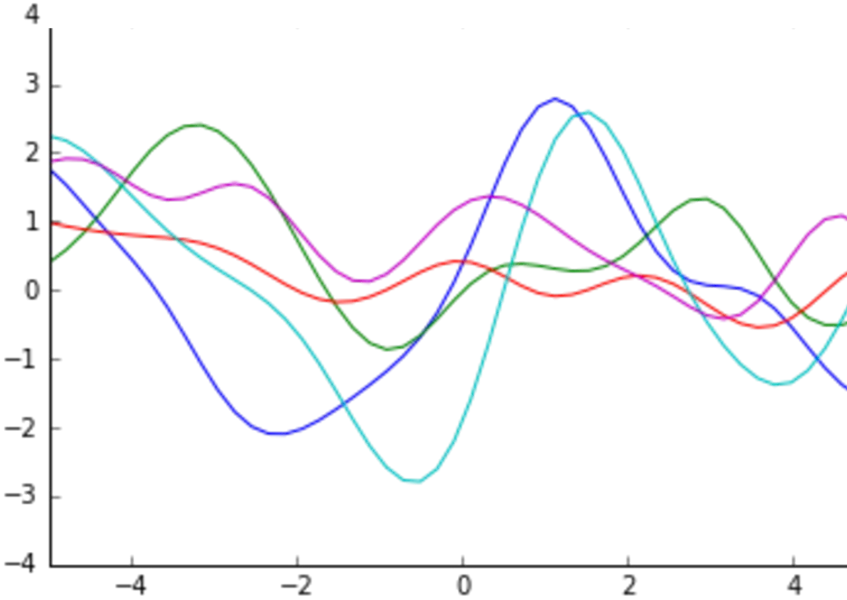


Over some restricted input space...

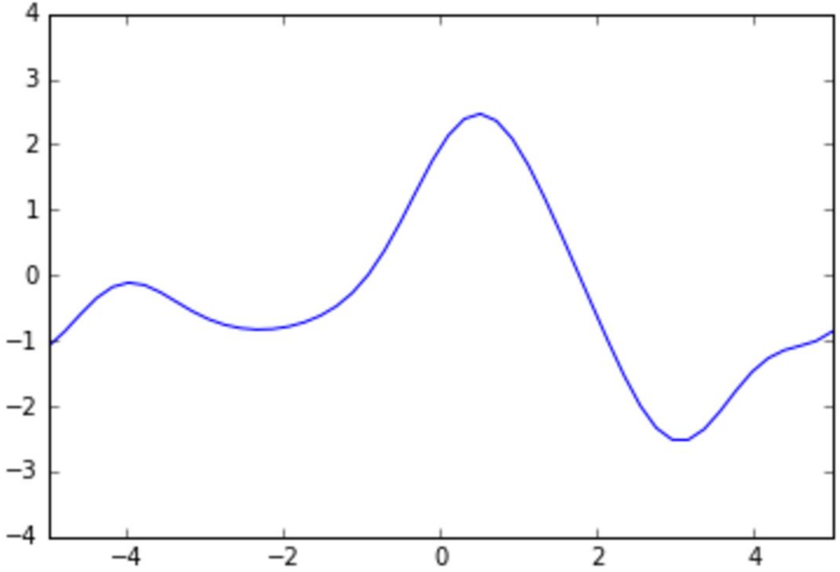
- Come up with a prior distribution over functions.
- Observe some data
- Come up with a posterior distribution over functions.
- Sample from that posterior distribution to get predictions for unobserved values of x

A sensible prior

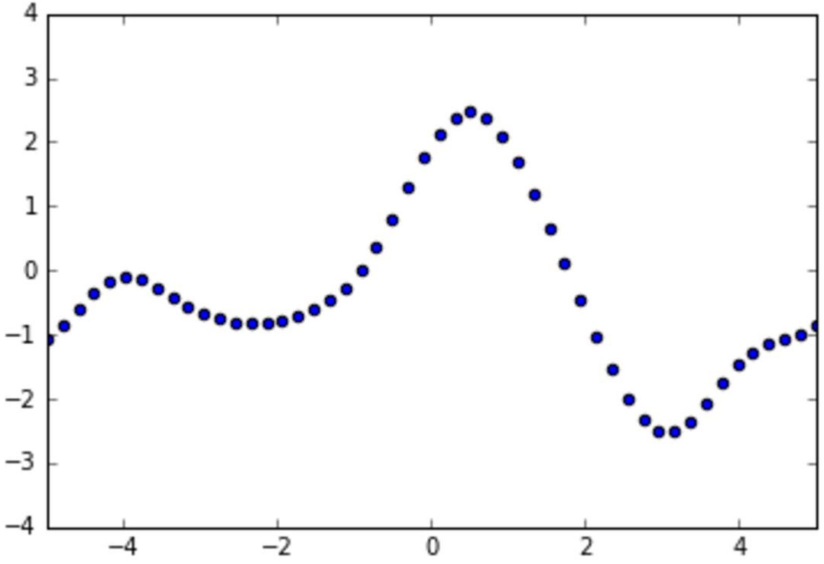
Input values that are close together should produce similar outputs



Functions as mappings of inputs to outputs



Functions as mappings of inputs to outputs



x = [-5.0, -4.8, -4.6, …, 4.6, 4.8, 5.0]
y = [-1.085, -0.862, -0.596, …, -1.081, -1.007, -0.863]

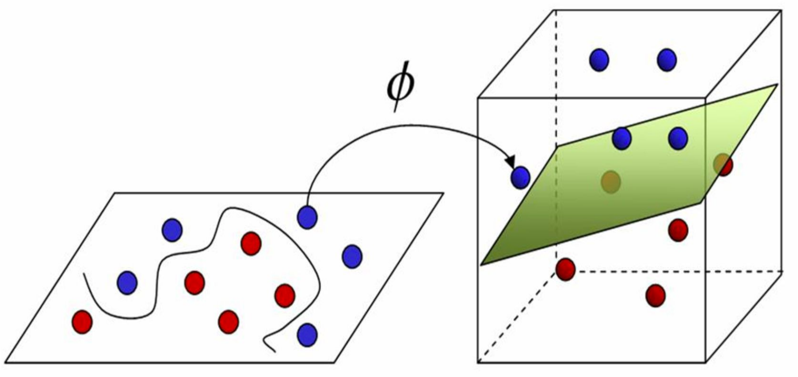
Kernel function

A kernel function is a function that outputs a measure of similarity between two data points.

Gaussian kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

An Aside: The “Kernel Trick”



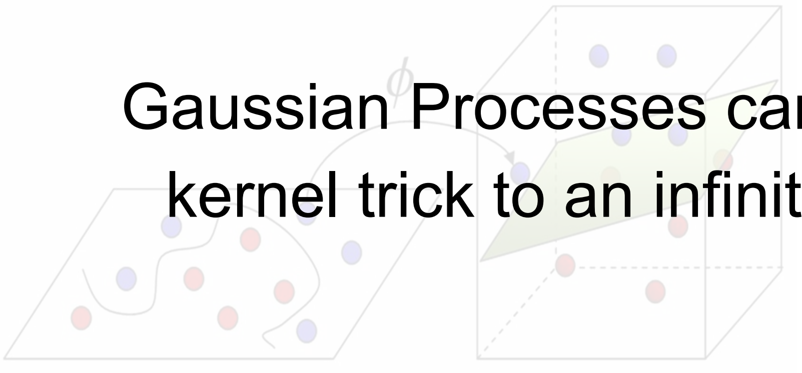
Explode the feature space to create a more flexible model

Rewrite algorithms in terms of dot products between examples

Note that the dot product of two vectors is a measure of their similarity

Replace this with a more general “kernel function” that measures their similarity without you ever having to compute the actual mapping in the higher dimensional space

An Aside: The “Kernel Trick”



Explode the feature space to create a more flexible model

Rewrite algorithms in terms of dot products between examples

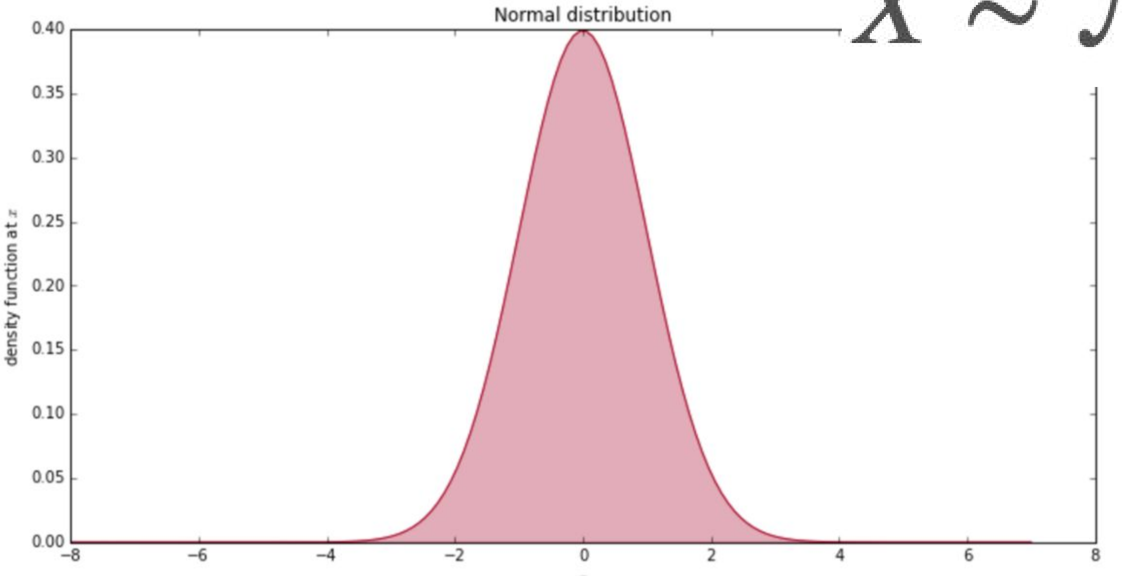
Note that the dot product of two vectors is a measure of their similarity

Replace this with a more general “kernel function” that measures their similarity without you ever having to compute the actual mapping in the higher dimensional space

Gaussian Processes can be thought of as applying the kernel trick to an infinite-dimensional feature space.

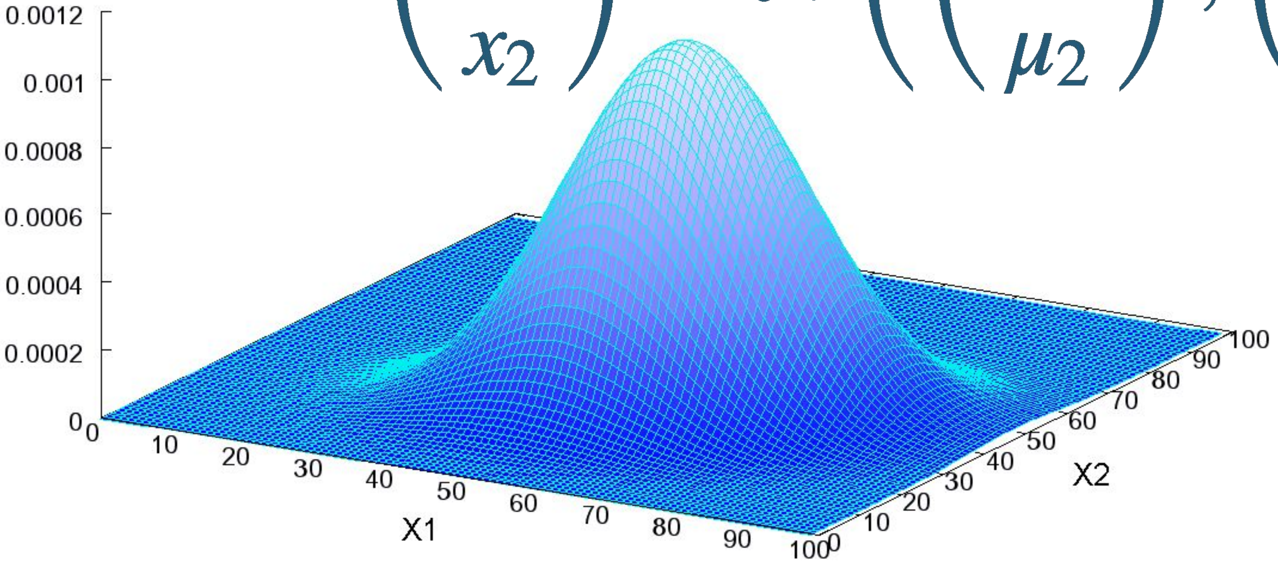
The Bell Curve aka Normal or Gaussian Distribution

$$X \sim \mathcal{N}(\mu, \sigma^2)$$



The Multivariate Gaussian Distribution

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix} \right)$$



The Multivariate Gaussian Distribution

Assume our f(x)'s are multivariate Gaussian distributed:

$$\begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \dots \\ f(x_n) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} m(x_1) \\ m(x_2) \\ m(x_3) \\ \dots \\ m(x_n) \end{pmatrix}, \begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & k(x_1,x_3) & \dots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & k(x_2,x_3) & \dots & k(x_2,x_n) \\ k(x_3,x_1) & k(x_3,x_2) & k(x_3,x_3) & \dots & k(x_3,x_n) \\ \dots & \dots & \dots & \dots & \dots \\ k(x_n,x_1) & k(x_n,x_2) & k(x_n,x_3) & \dots & k(x_n,x_n) \end{pmatrix}\right)$$

The Multivariate Gaussian Distribution

With test points X_test = [1, 2, 3, 4]

$$\begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} k(1,1) & k(1,2) & k(1,3) & k(1,4) \\ k(2,1) & k(2,2) & k(2,3) & k(2,4) \\ k(3,1) & k(3,2) & k(3,3) & k(3,4) \\ k(4,1) & k(4,2) & k(4,3) & k(4,4) \end{pmatrix} \right)$$

The Multivariate Gaussian Distribution

We are assuming our function outputs are jointly Gaussian, and now we have observed some of them...

Vector of training points (observed data)

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} K & K_* \\ K_*^T & K_{**} \end{pmatrix} \right)$$

Vector of test points

Conditional distribution

With this assumption

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} K & K_* \\ K_*^T & K_{**} \end{pmatrix}\right)$$

We can get the conditional distribution of the test outputs given the training outputs

$$p(f_* | f)$$

The posterior distribution

Many pages of matrix algebra later…

$$f_* \sim \mathcal{N}(\mu_*, \Sigma_*)$$

We have the mean and covariance matrix for the posterior distribution and now we can sample from it!

Sampling from the posterior

Recall that when we have a univariate normal $x \sim \mathcal{N}(\mu, \sigma^2)$

This can be expressed in terms of the standard normal

$$x \sim \mu + \sigma(\mathcal{N}(0, 1))$$

We need the same idea for multivariate normals $f_* \sim \mathcal{N}(\mu_*, \Sigma_*)$

$$f_* \sim \mu + B\mathcal{N}(0, I)$$

Where $BB^T = \Sigma_*$

Sampling from the posterior

Recall that when we have a univariate normal $x \sim \mathcal{N}(\mu, \sigma^2)$

This can be expressed in terms of the standard normal

$$x \sim \mu + \sigma(\mathcal{N}(0, 1))$$

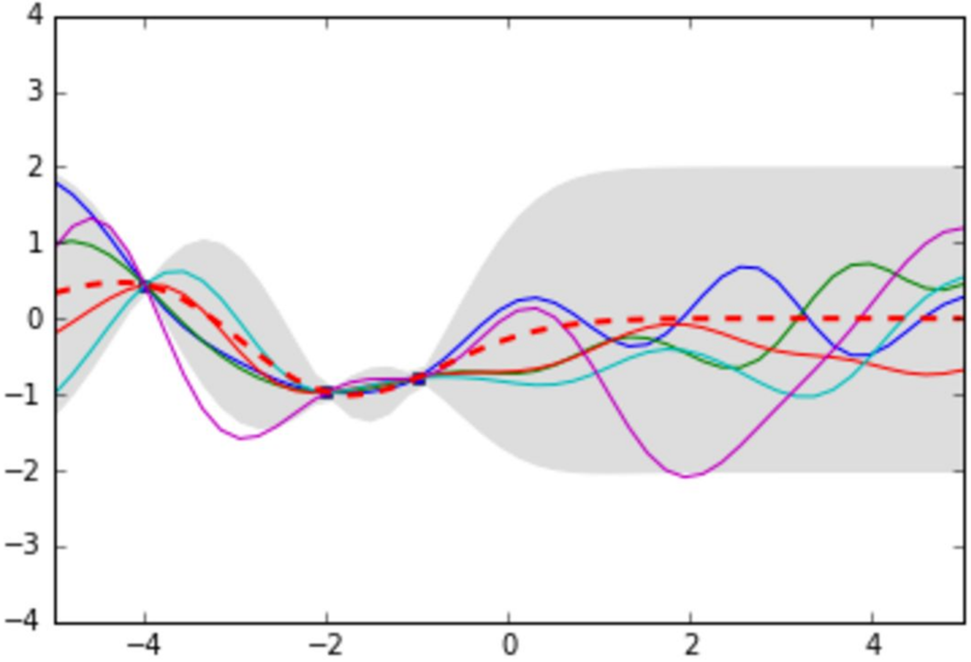
We need the same idea for multivariate normals $f_* \sim \mathcal{N}(\mu_*, \Sigma_*)$

$$f_* \sim \mu + B\mathcal{N}(0, I)$$

Where $BB^T = \Sigma_*$

Cholesky decomposition

Samples from the posterior

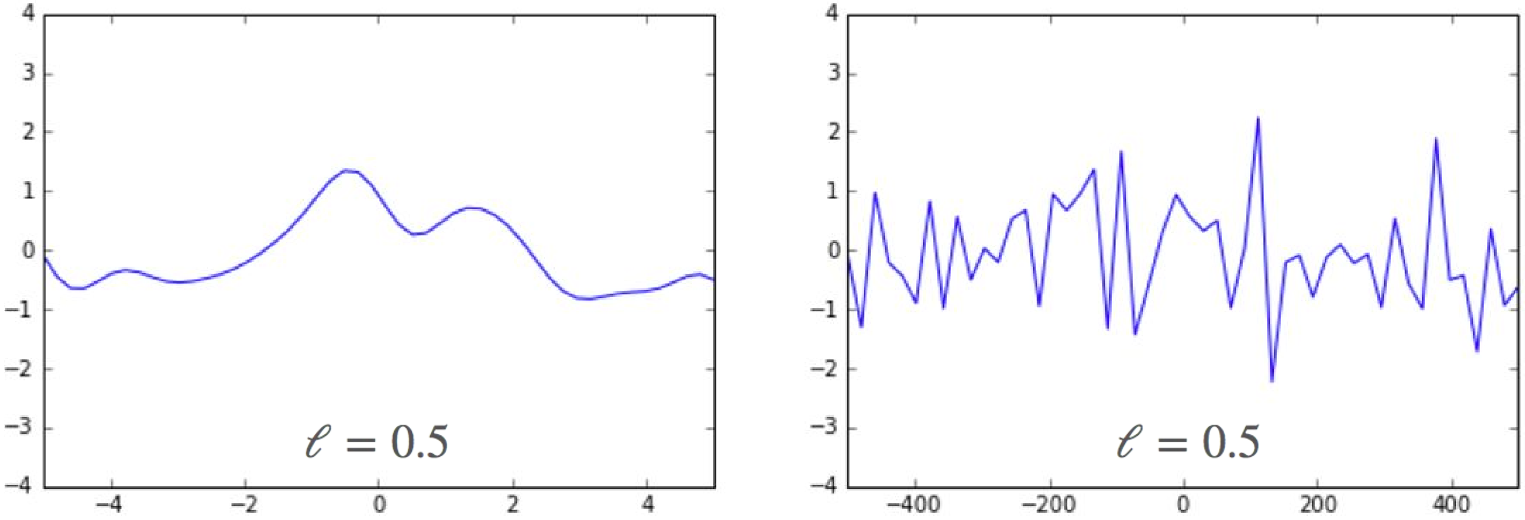


Kernel parameters

How similar are the numbers 3 and 4?

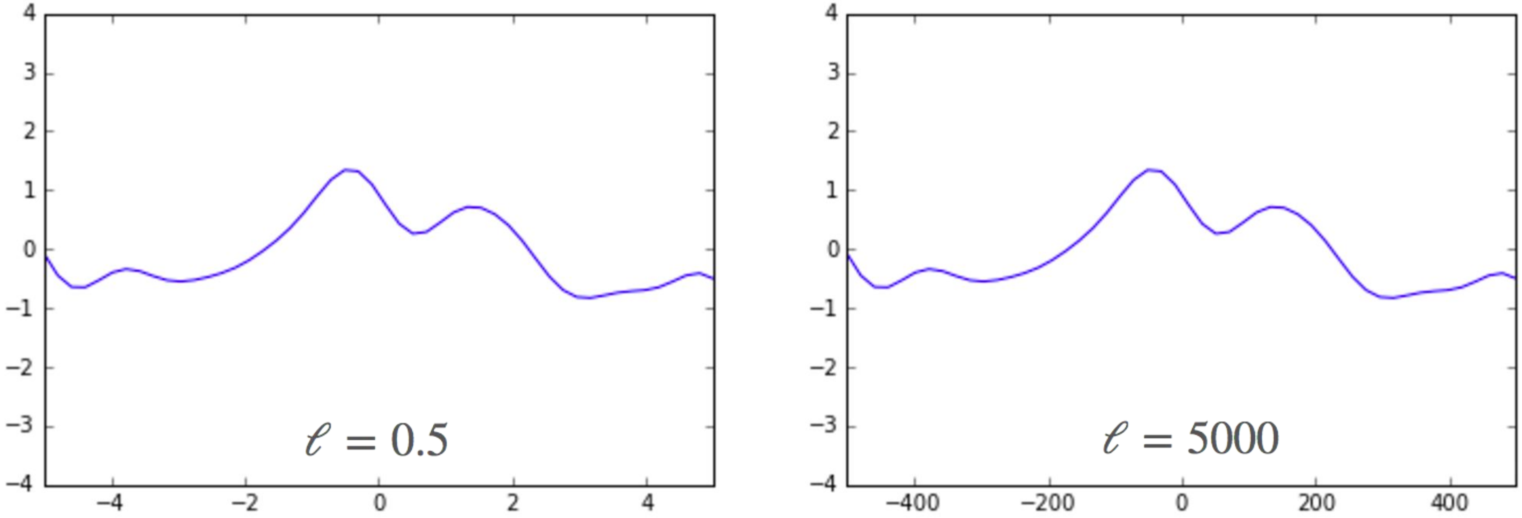
$$\begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} k(1,1) & k(1,2) & k(1,3) & k(1,4) \\ k(2,1) & k(2,2) & k(2,3) & k(2,4) \\ k(3,1) & k(3,2) & k(3,3) & k(3,4) \\ k(4,1) & k(4,2) & k(4,3) & k(4,4) \end{pmatrix} \right)$$

Effect of the length scale parameter



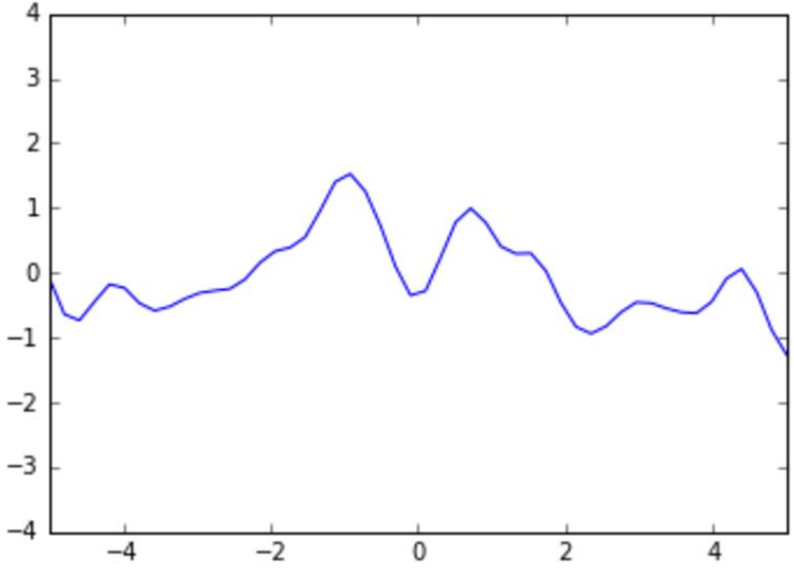
The distance you have to move in input space before the function value can change significantly

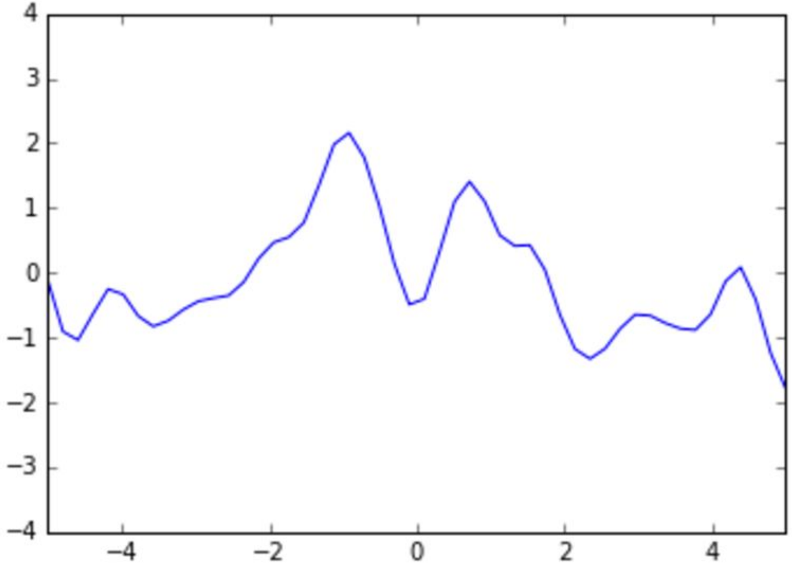
Effect of the length scale parameter



The distance you have to move in input space before the function value can change significantly

Effect of the vertical scale parameter





Code

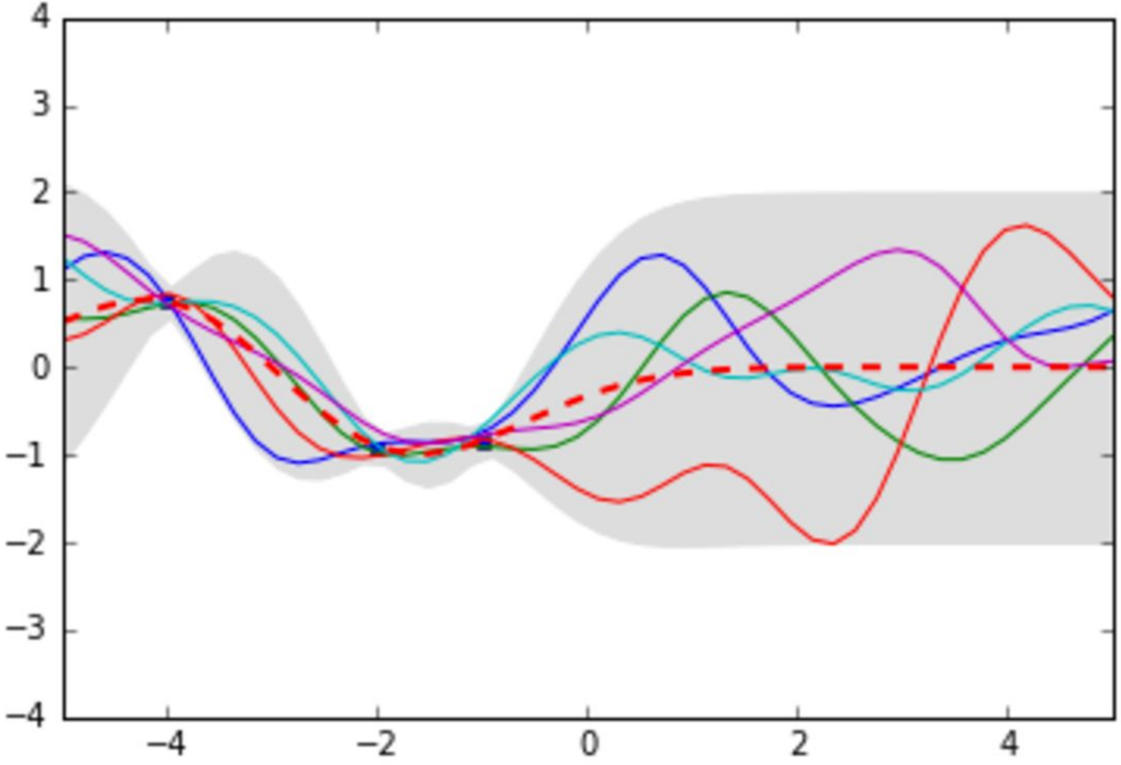
```
def kernel(a, b, l_scale, v_scale = 1):
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a, b.T)
    return v_scale * np.exp(-.5 * (1/l_scale) * sqdist)

K = kernel(Xtrain, Xtrain, l_scale, v_scale)
K_s = kernel(Xtrain, Xtest, l_scale, v_scale)
K_ss = kernel(Xtest, Xtest, l_scale, v_scale)

L = jitter_chol(K + noise*np.eye(Xtrain.shape[0]))
Lk = np.linalg.solve(L, K_s)
mu = np.dot(Lk.T, np.linalg.solve(L, ytrain)).reshape((n,))
L_ = jitter_chol(K_ss - np.dot(Lk.T, Lk))
f_post = mu.reshape(-1,1) + np.dot(L_, np.random.normal(size=(n,5)))

stdv = np.sqrt(np.diag(K_ss) - np.sum(Lk**2, axis=0))
pl.plot(Xtrain, ytrain, 'bs', ms=4)
pl.plot(Xtest, f_post)
pl.gca().fill_between(Xtest.flat, mu-2*stdv, mu+2*stdv, color="#dddddd")
pl.plot(Xtest, mu, 'r--', lw=2)
```

Samples from the posterior - noisy data



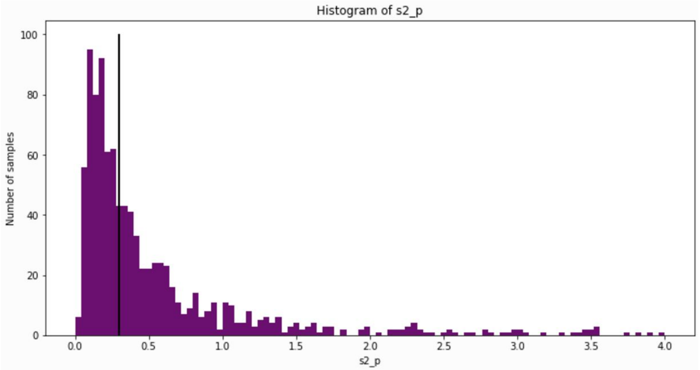
Optimizing the hyperparameters

Maximum likelihood, as in scikit-learn:

```
# Instanciate a Gaussian Process model
kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)

# Fit to data using Maximum Likelihood Estimation of the parameters
gp.fit(X, y)
```

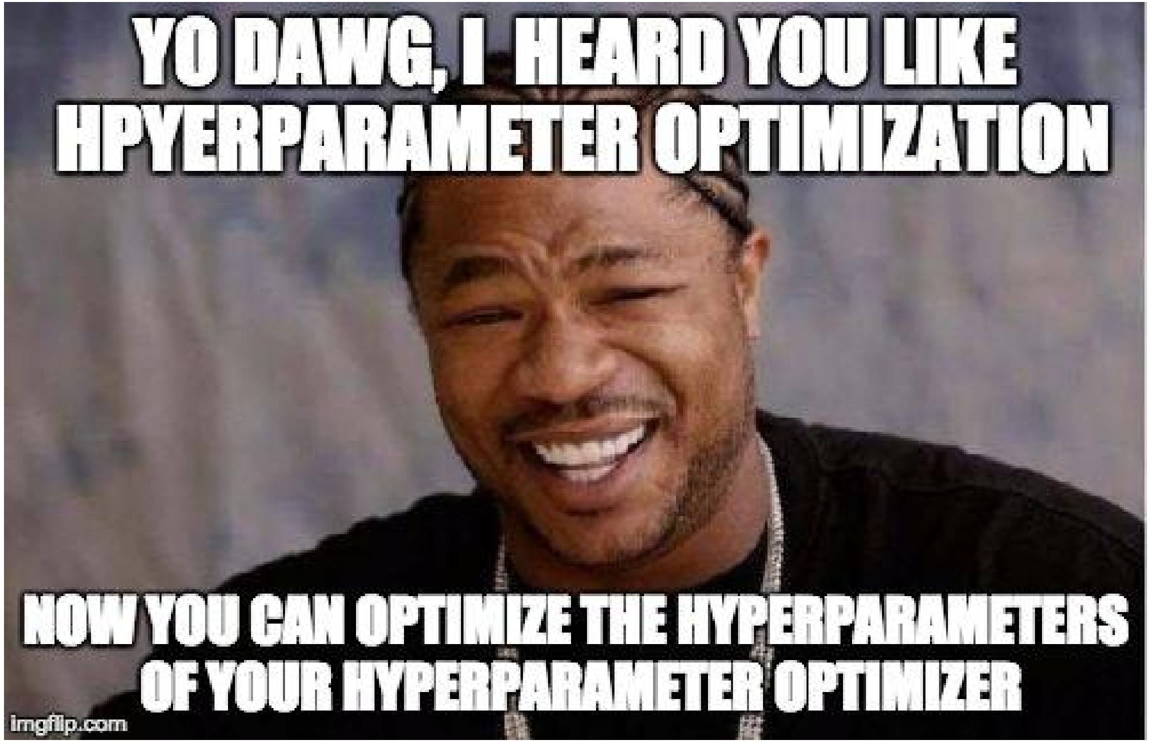
MCMC



Bayesian Optimization

Gaussian Processes can be used to optimize the hyperparameters of other models such as Neural Networks.

This is how Bayesian Optimization works.


YO DAWG, I HEARD YOU LIKE
HPYERPARAMETER OPTIMIZATION
NOW YOU CAN OPTIMIZE THE HYPERPARAMETERS
OF YOUR HYPERPARAMETER OPTIMIZER
imgflip.com

Tuning Neural Networks is hard

How do you figure out the right values for all these things?

- Number of hidden units
- Number of layers
- Weight penalty
- Learning rate
- Whether to use dropout

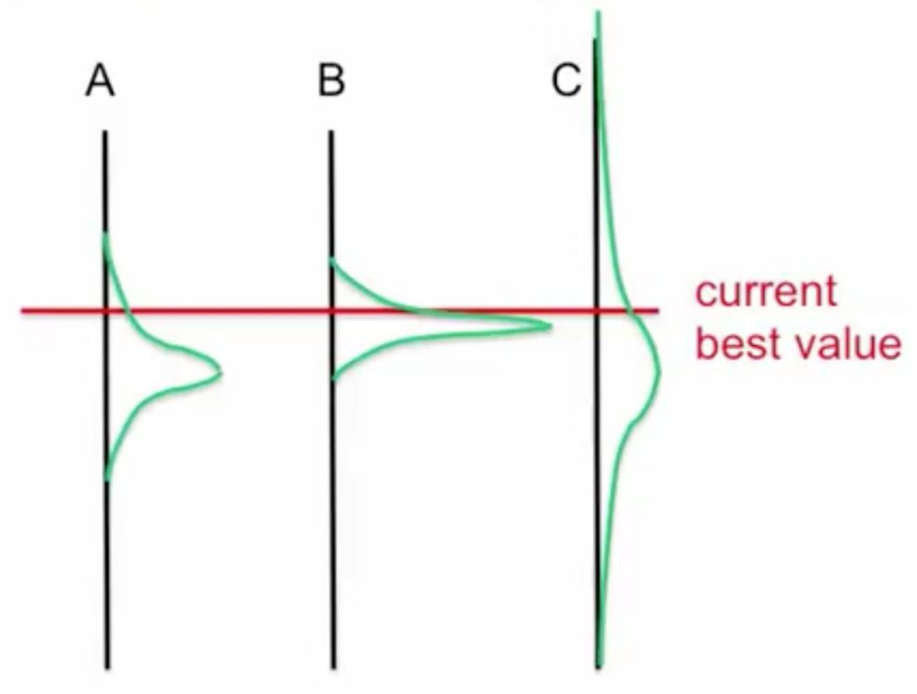
Traditional approaches

- Grad student descent :)
- Grid search
  - List all possible values of each parameter and then systematically try all the combinations
- Random search
  - Randomly sample combinations of parameter values (better than grid search)

Bayesian Optimization

- Exploration of a single combination of parameters is relatively expensive
- We'd like to choose the next combination to try as intelligently as possible
- Using a GP we take the output from previous runs of the NN as training points and then come up with the mean and variance of unobserved areas of the parameter space
- The next combination to try will be the one with the highest Expected Improvement (EI)
- This idea originally came from the world of gold mining and was called kriging

Bayesian Optimization



Bayesian Optimization

MOE

Σ SIGOPT

Amplify your research

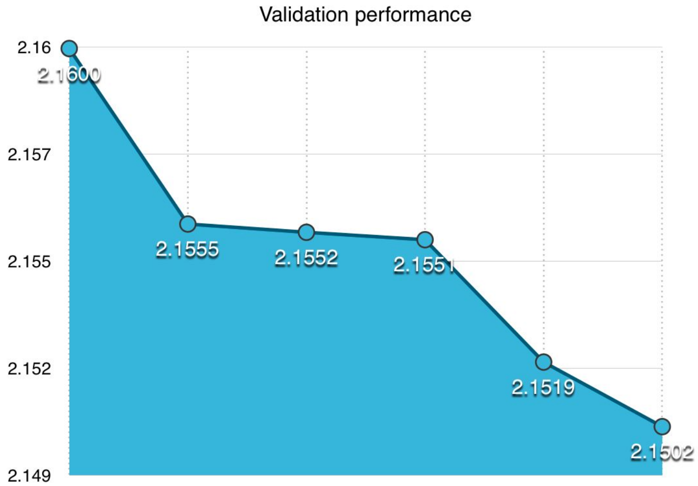
SigOpt takes any research pipeline and tunes it, right in place, boosting your business objectives. Our cloud-based ensemble of optimization algorithms is proven and seamless to deploy.

Spearmint

Spearmint is a package to perform Bayesian optimization according to the algorithms outlined in the paper:

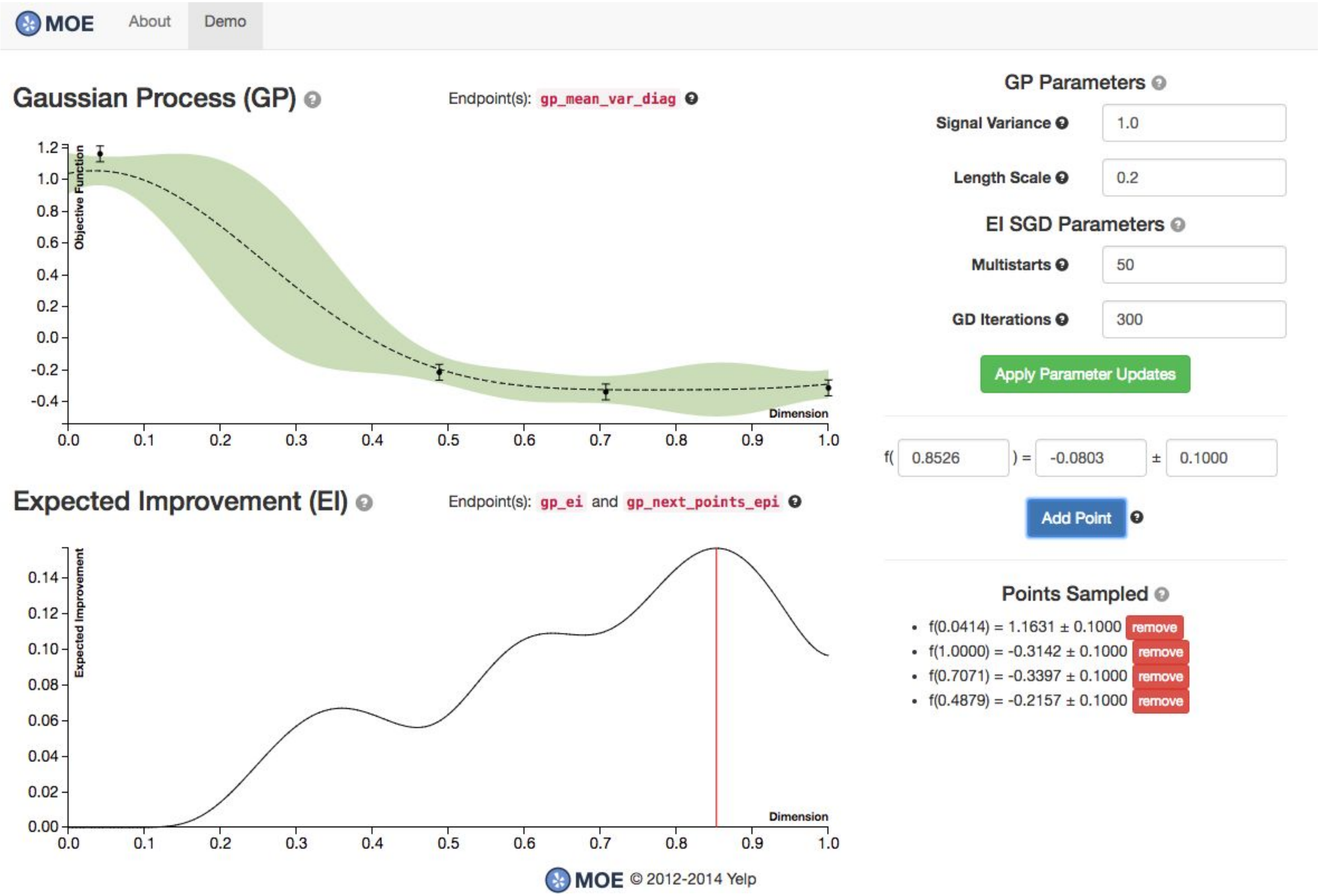
Practical Bayesian Optimization of Machine Learning Algorithms
Jasper Snoek, Hugo Larochelle and Ryan P. Adams
Advances in Neural Information Processing Systems, 2012

Bayesian Optimization



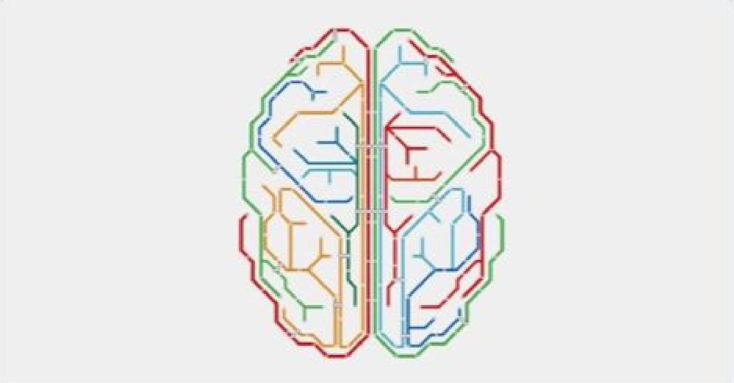
Completed jobs	1	7	16	17	18	39
Elapsed time	1021	8003	12742	1623	1983	31561
Number of layers	1	3	3	2	3	2
Hidden units	64	256	256	256	256	256
Learning rate	0.01	0.1	0.01	0.01	0.021169	0.025994
Input dropout	0	0.5	0.5	0.462897	0.5	0.423678
Hidden dropout	0	0	0	0.024181	0.089289	0.091693
Weight decay	0	0	0.001903	0.003372	0.00019	0.00238

From https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/


MOE
About
Demo
Gaussian Process (GP)
Endpoint(s): gp_mean_var_diag
Objective Function
Dimension
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Expected Improvement (EI)
Endpoint(s): gp_ei and gp_next_points_epi
Expected Improvement
Dimension
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
GP Parameters
Signal Variance
1.0
Length Scale
0.2
EI SGD Parameters
Multistarts
50
GD Iterations
300
Apply Parameter Updates
f(0.8526) = -0.0803 ± 0.1000
Add Point
Points Sampled
f(0.0414) = 1.1631 ± 0.1000 remove
f(1.0000) = -0.3142 ± 0.1000 remove
f(0.7071) = -0.3397 ± 0.1000 remove
f(0.4879) = -0.2157 ± 0.1000 remove
MOE © 2012-2014 Yelp

Andrew Ng
@AndrewYNg

Wired on Gaussian Processes. IMO they're fine to use, but just hard scale to big data. wired.com/2017/02/ai-lea...



AI Is About to Learn More Like Humans—with a Little Uncertainty

Neural networks are all the rage right now. But they're still flawed. So top tech companies are looking at new forms of AI better at handling uncertainty.

Pros & Cons of GPs

Pros

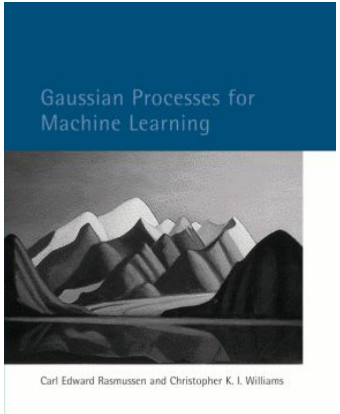
- Predictions can interpolate observations
- Predictions are probabilistic so that one can compute confidence intervals
- Flexibility: you can use different kernels

Cons

- They lose efficiency in high dimensional spaces – namely when the number of features exceeds a few dozens.

Resources

- Rasmussen & Williams book: http://www.gaussianprocess.org/gpml/
- Nando de Freitas' lectures on YouTube https://www.youtube.com/watch?v=4vGiHC35j9s
- Chapter 15 of Kevin Murphy's ML book
- PyMC docs on GPs http://pymc-devs.github.io/pymc3/notebooks/GP-introduction.html
- Geoff Hinton's talk on Bayesian Optimization of NNs https://www.youtube.com/watch?v=con_ONbhD2I



Resources

- Bayesion Optimization in Scikit Learn
  https://thuijskens.github.io/2016/12/29/bayesian-optimisation/
- Bayesian Optimization paper by Snoek, Larochelle & Adams:
  http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf
- Gaussian Processes for Big Data by Hensman, Fusi & Lawrence:
  http://www.auai.org/uai2013/prints/papers/244.pdf
- My blog post :)
  http://katbailey.github.io/post/gaussian-processes-for-dummies/

Thanks!

@katherinebailey