APS1070

Foundations of Data Analytics and Machine Learning

Summer 2020

Wed July 12 / Week 12:

- *Deep Learning*
- *Naïve Bayes + Bayesian Optimization*
- *Review / Exercises*

Jason Riordon, PhD

News

- Project 4 due **Sunday, Aug 2 @ 11pm**
- Final Quiz next week: **Tuesday, Aug 4 @ 1pm** to **Wednesday, Aug 5 @ 11:59pm**
- Reminder – Course Evaluations!
- Last lecture!

Slide Attribution

These slides contain materials from various sources. Special thanks to Roger Grosse, Scott Sanner and Marc Deisenroth.

Deep Learning

Inspiration: The Brain

- Our brain has $\sim 10^{11}$ neurons, each of which communicates (is connected) to $\sim 10^{4}$ other neurons

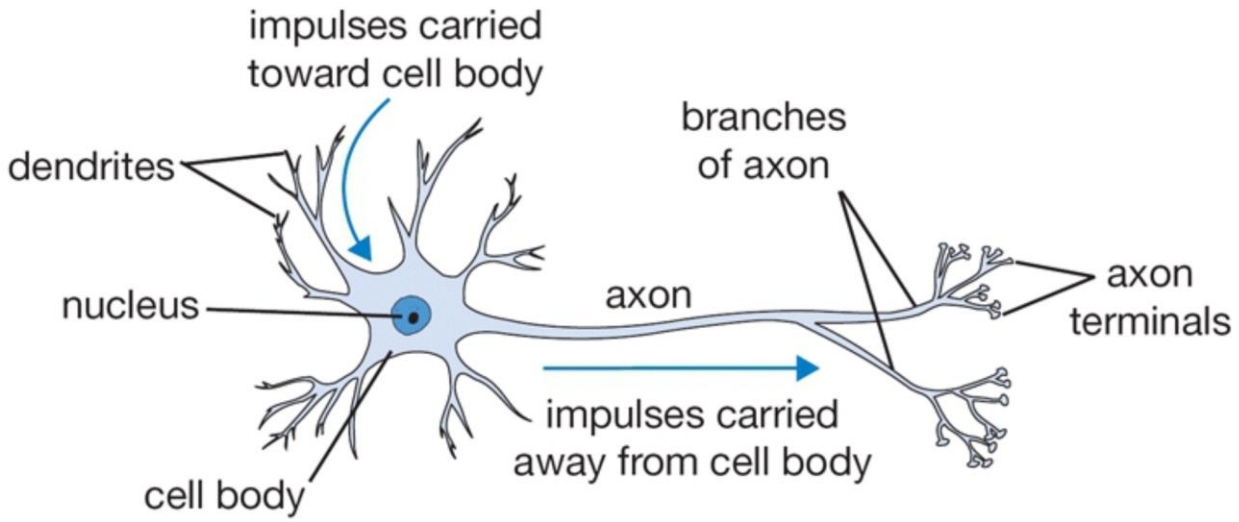
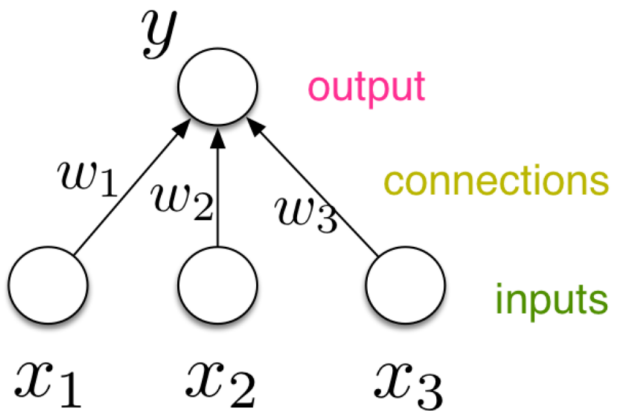


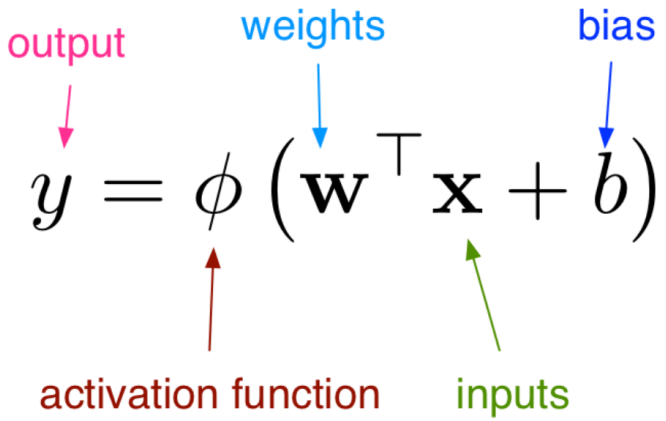
Figure: The basic computational unit of the brain: Neuron

[Pic credit: http://cs231n.github.io/neural-networks-1/]

Inspiration: The Brain

- For neural nets, we use a much simpler model neuron, or unit:





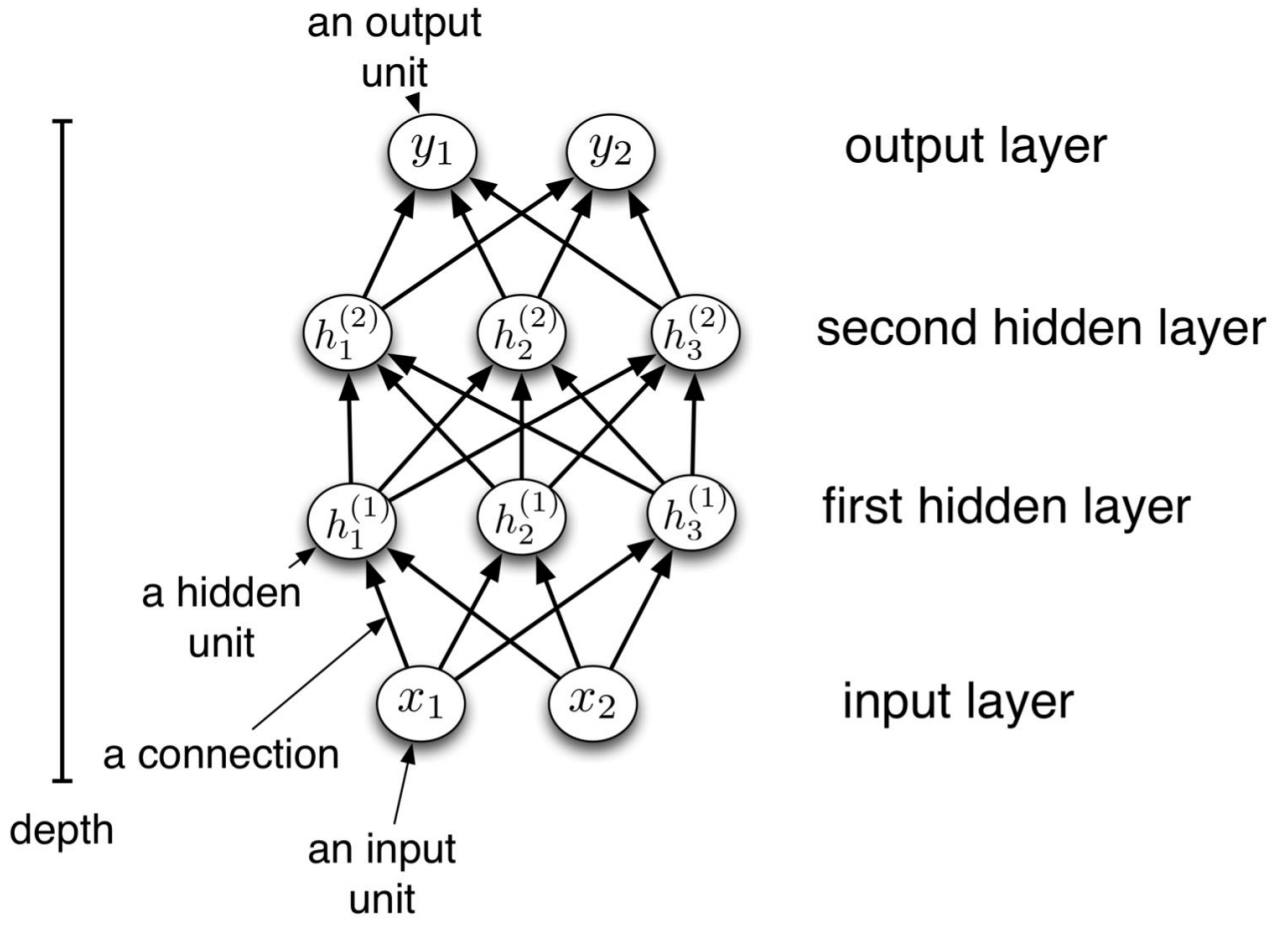
- Compare with logistic regression:

$$y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$

- By throwing together lots of these incredibly simplistic neuron-like processing units, we can do some powerful computations!

Multilayer Perceptron

- We can connect lots of units together into a directed acyclic graph.
- This gives a feed-forward neural network. That's in contrast to recurrent neural networks, which can have cycles.
- Typically, units are grouped together into layers.

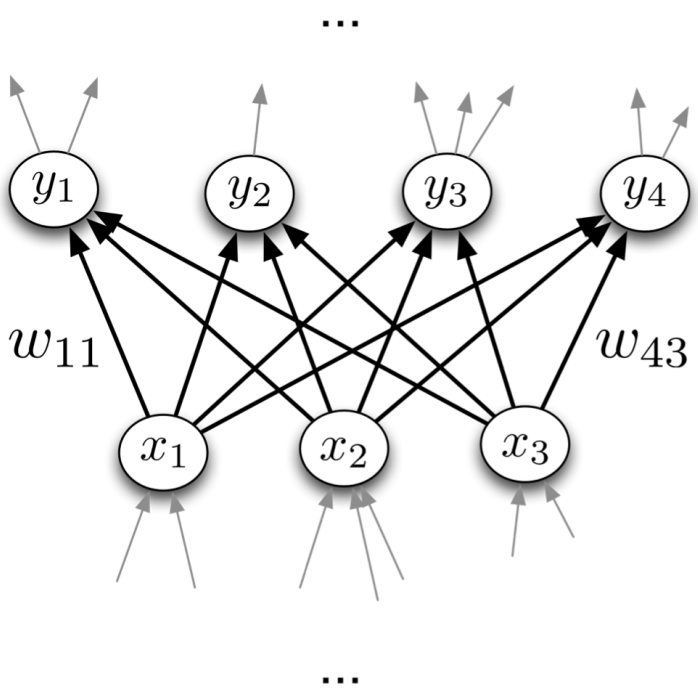


Multilayer Perceptron

- Each layer connects N input units to M output units.
- In the simplest case, all input units are connected to all output units. We call this a fully connected layer. We'll consider other layer types later.
- Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network.

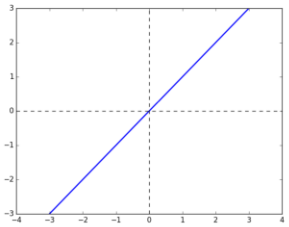
- Recall from softmax regression: this means we need an $M \times N$ weight matrix.
- The output units are a function of the input units:

$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$



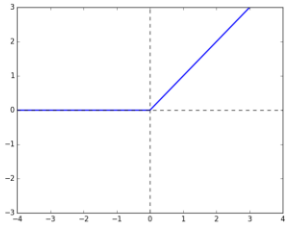
Multilayer Perceptron

Some activation functions:



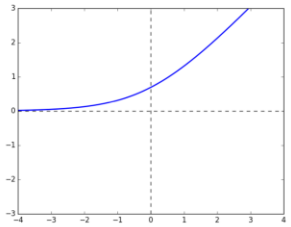
Linear

$y = z$



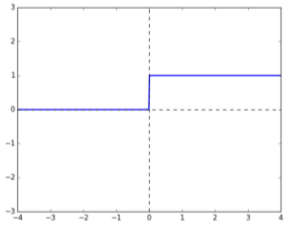
Rectified Linear Unit (ReLU)

$y = \max(0, z)$



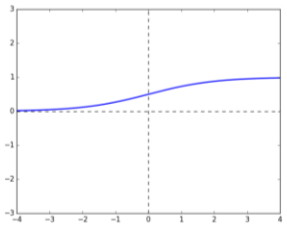
Soft ReLU

$y = \log 1 + e^z$



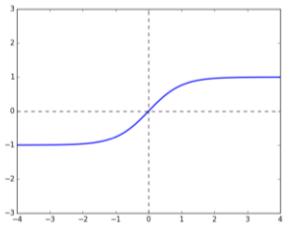
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



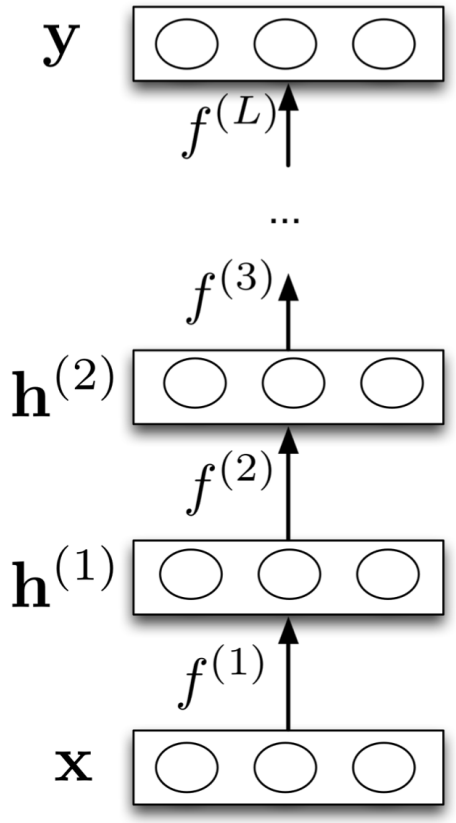
Hyperbolic Tangent (tanh)

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Multilayer Perceptron

- Each layer computes a function, so the network computes a composition of functions:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x})$$
$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)})$$
$$\vdots$$
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$



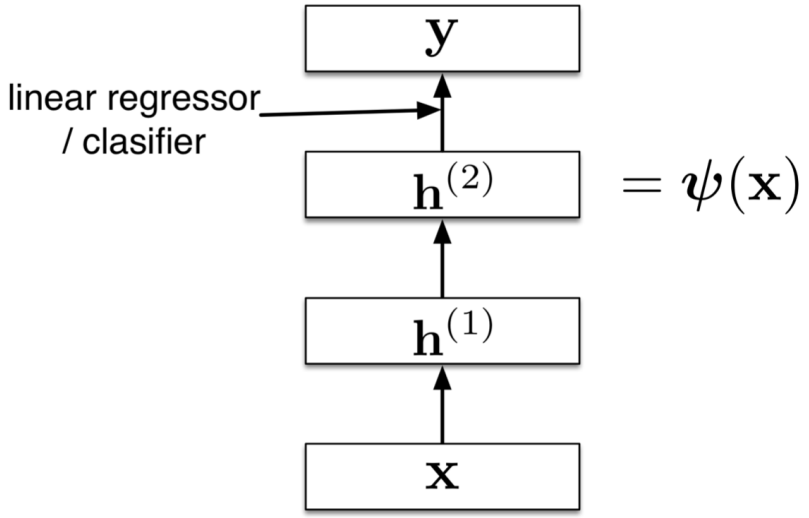
- Or more simply:

$$\mathbf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).$$

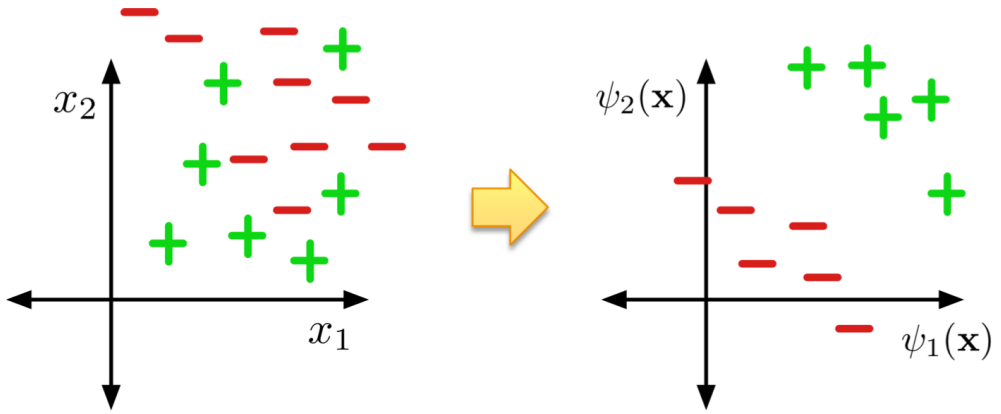
- Neural nets provide modularity: we can implement each layer's computations as a black box.

Multilayer Perceptron

- Neural nets can be viewed as a way of learning features:



- The goal:

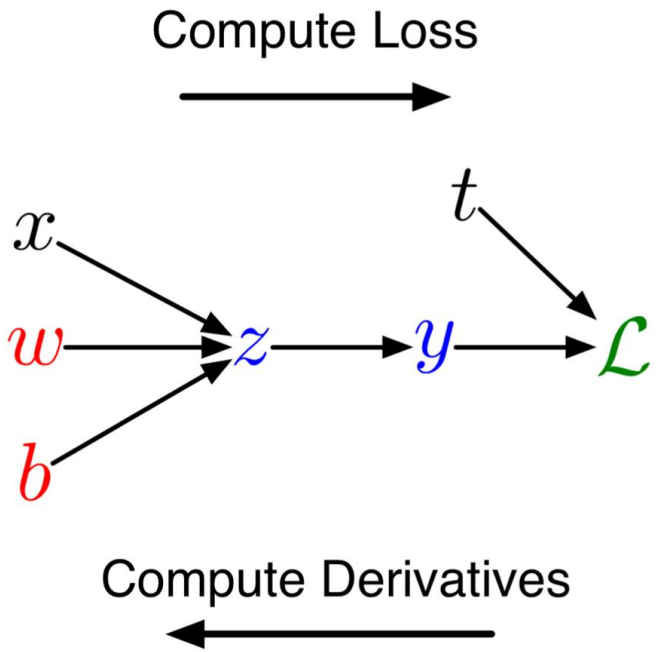


Expressive Power

- Multilayer feed-forward neural nets with *nonlinear* activation functions are universal function approximators: they can approximate any function arbitrarily well.
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though ReLU is "almost" linear, it's nonlinear enough!

How to find the weights (Backpropagation)

- We can diagram out the computations using a computation graph.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.



How to find the weights (Backpropagation)

- We can diagram out the computations using a computation graph.

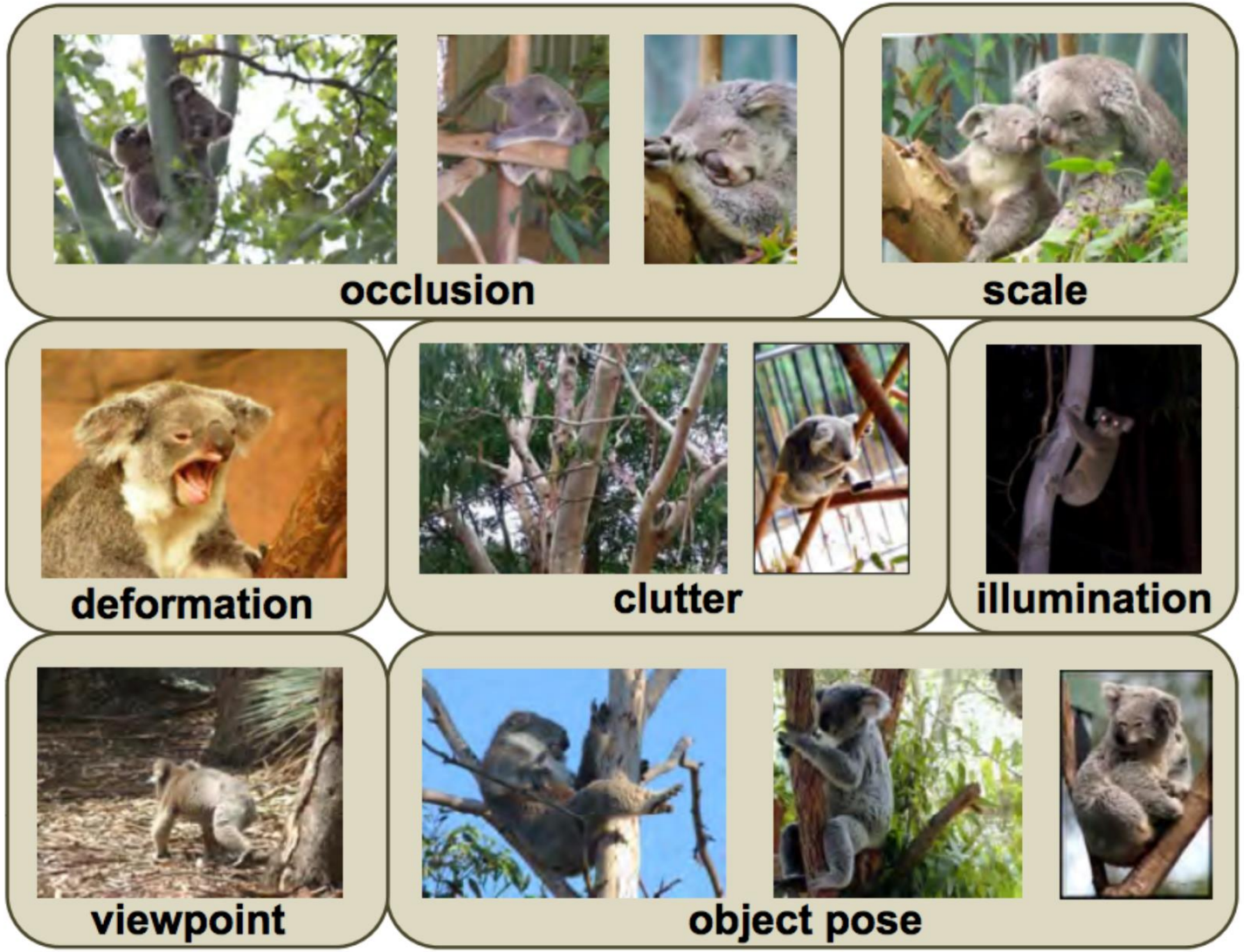
We use gradient descent!
The gradient are required to update the weights!

Compute Derivatives

Neural Nets for Visual Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it
- Why is it difficult?

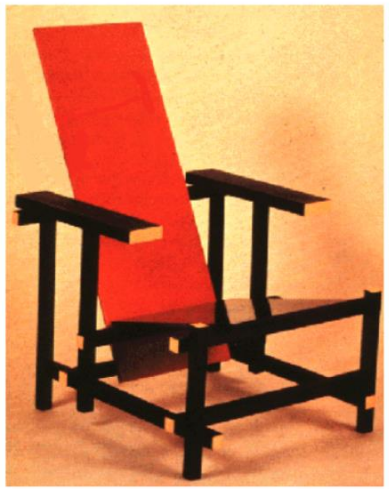
- Difficult scene conditions



[From: Grauman & Leibe]

- Huge within-class variations.



[Pic from: S. Lazebnik]

- Tons of classes

[Biederman]

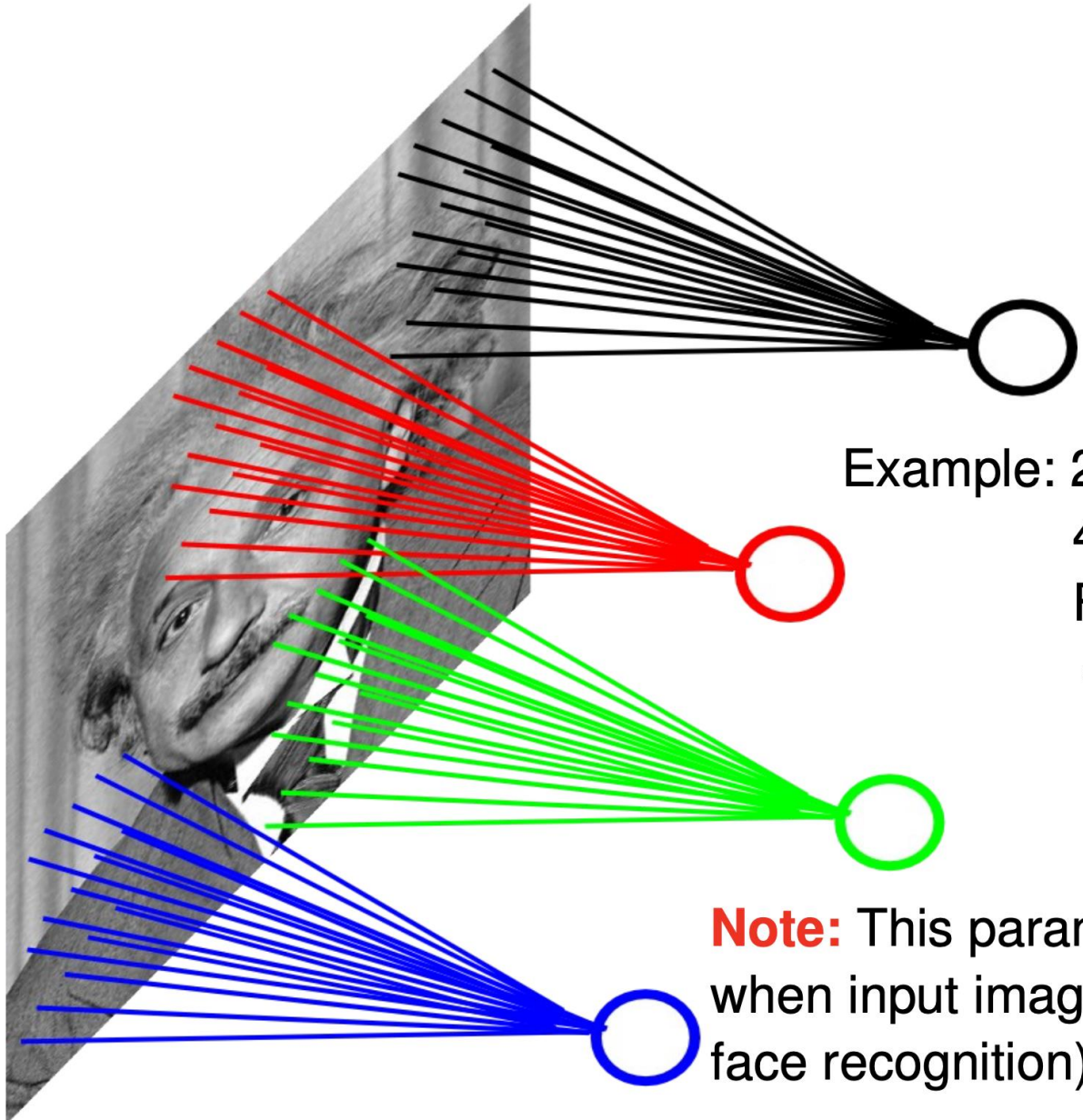
Neural Nets for Object Recognition

- People are very good at recognizing object
  - Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:
  - Segmentation: Real scenes are cluttered
  - Invariances: We are very good at ignoring all sorts of variations that do not affect class
  - Deformations: Natural object classes allow variations (faces, letters, chairs)
  - A huge amount of computation is required

How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional
- How many parameters do I have?
- Prohibitive to have fully-connected layers
- What can we do?
- We can use a locally connected layer

Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

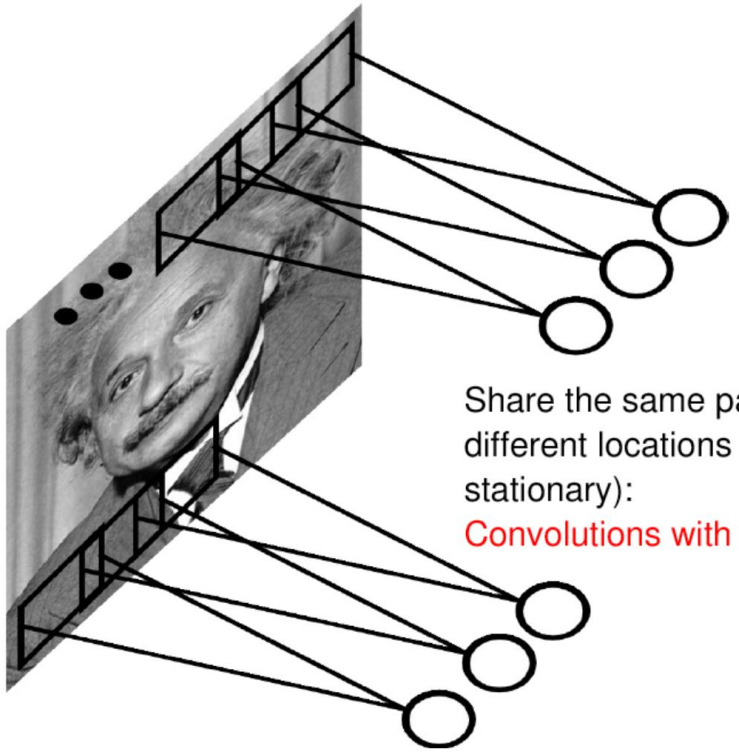
Note: This parameterization is good when input image is registered (e.g., face recognition).

The Invariance Problem

- Our perceptual systems are very good at dealing with invariances
  - translation, rotation, scaling
  - deformation, contrast, lighting
- We are so good at this that its hard to appreciate how difficult it is
  - Its one of the main difficulties in making computers perceive
  - We still don't have generally accepted solutions

Convolutional Neural Net

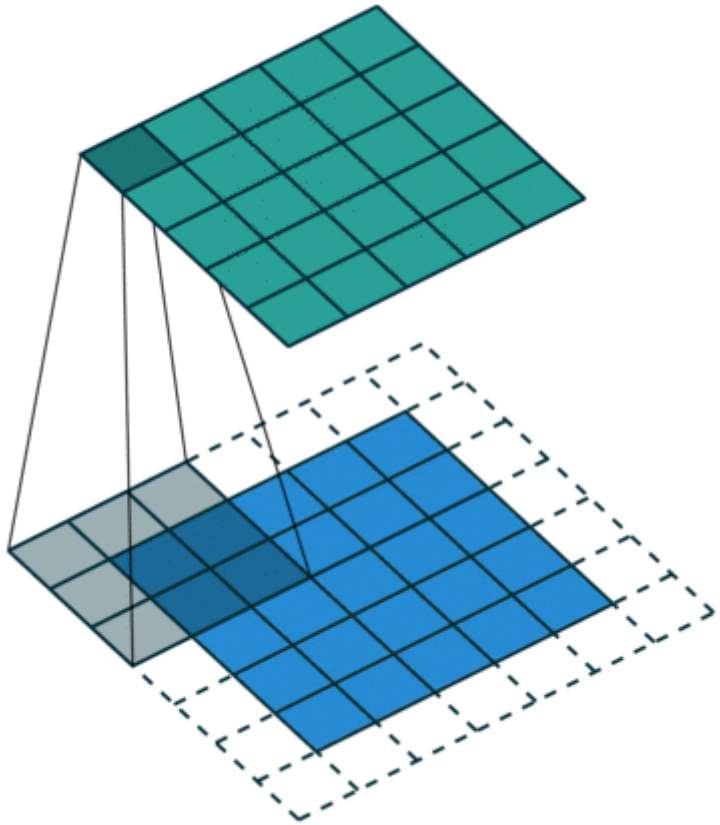
- Idea: statistics are similar at different locations (Lecun 1998)
- Connect each hidden unit to a small input patch and share the weight across space
- This is called a convolution layer and the network is a convolutional network



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

36
Ranzato

Convolution



Convolution

The thing we convolve by is called a kernel, or filter.

What does this convolution kernel do?

$*$

0	1	0
1	4	1
0	1	0

Convolution

What does this convolution kernel do?

$*$

0	-1	0
-1	8	-1
0	-1	0

Convolution

What does this convolution kernel do?

$*$

0	-1	0
-1	4	-1
0	-1	0

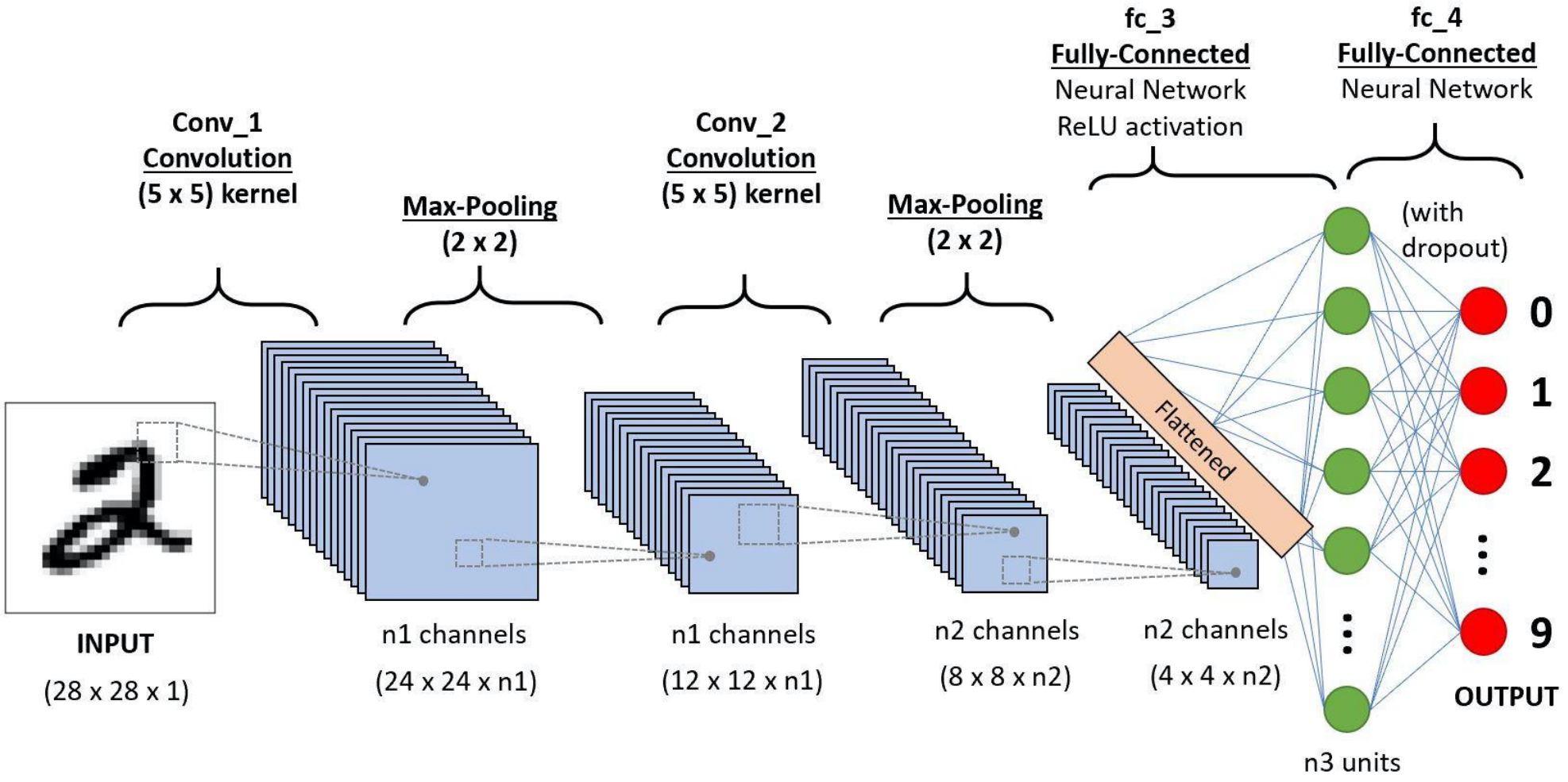
Convolution

What does this convolution kernel do?

$*$

1	0	-1
2	0	-2
1	0	-1

A CNN sequence to classify handwritten digits



Naïve Bayes + Bayesian Optimization

Bayes' Theorem

- We have a dataset of Spam and Not-spam emails.
- We look at Spam emails, 90% of them has the word "WIN".
- Now, if a new email has the word "WIN" what is the probability of that email being a spam?
- We don't have enough information!

Bayes' Theorem

Probability of "A" given "B"

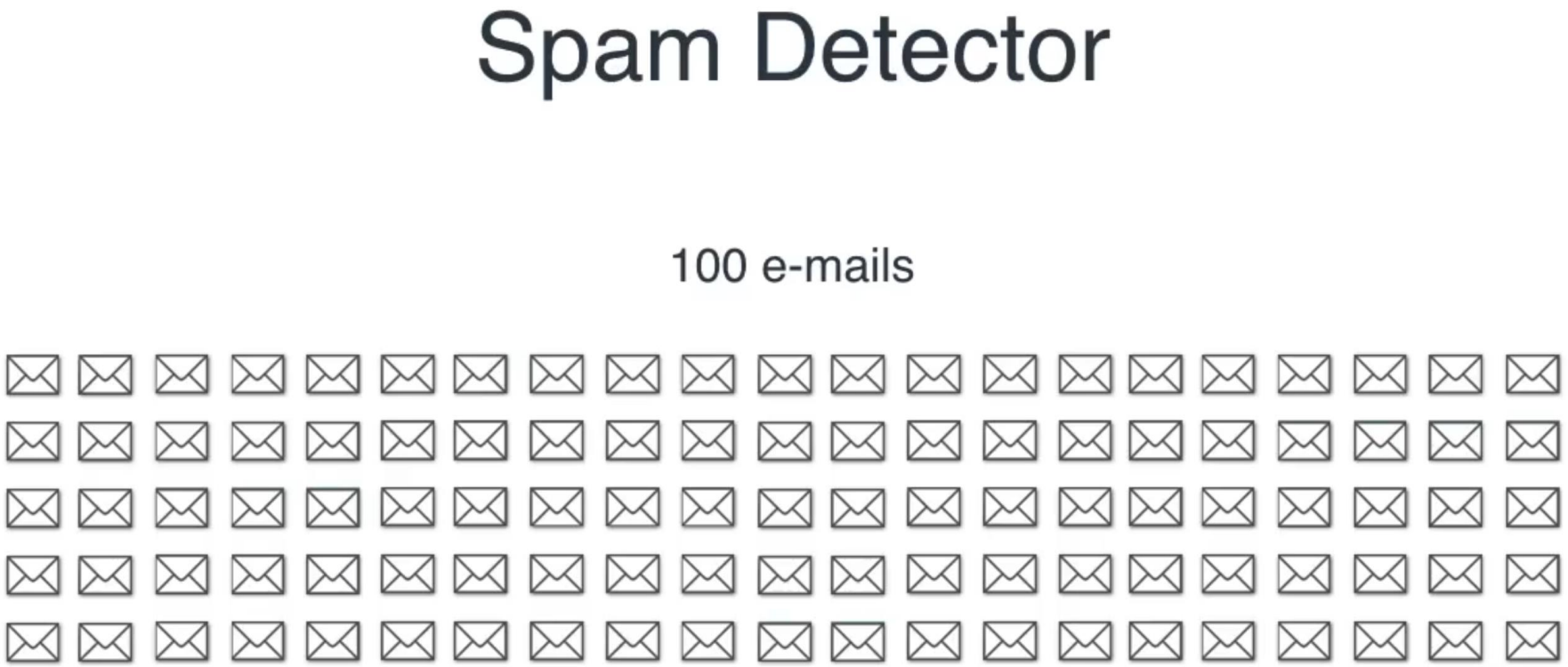
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes' Theorem

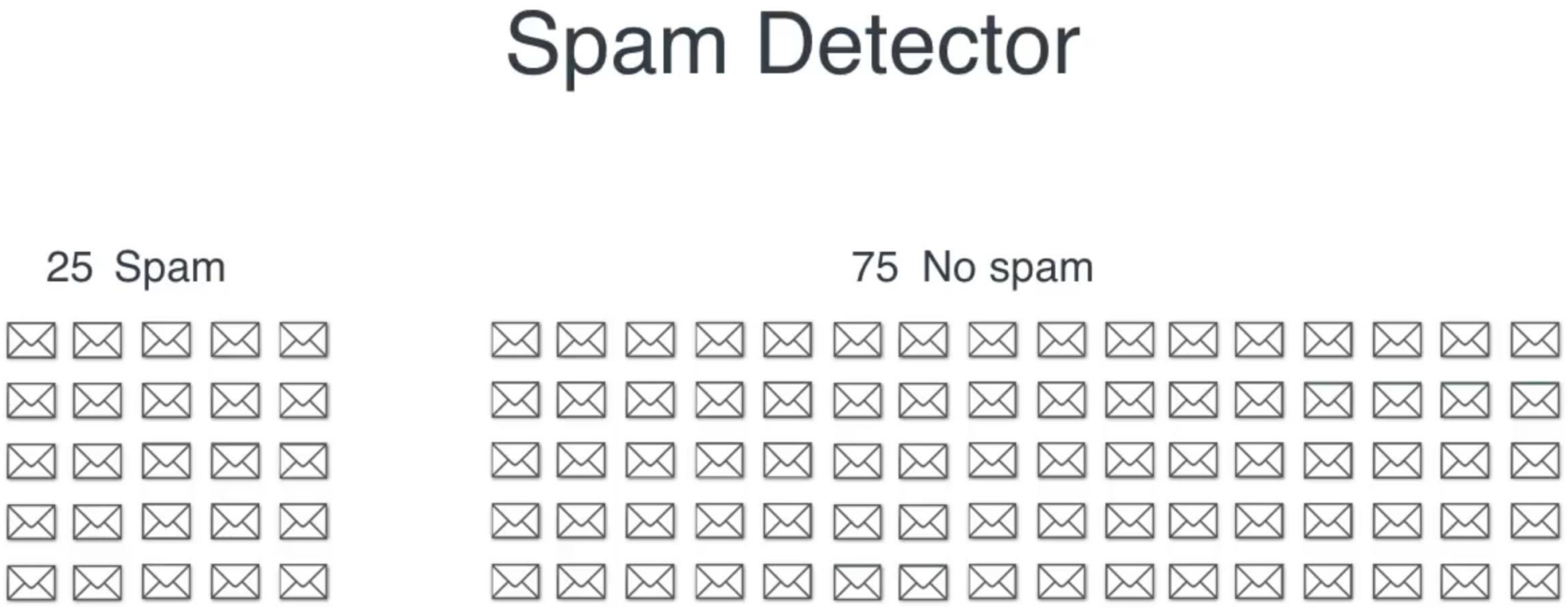
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

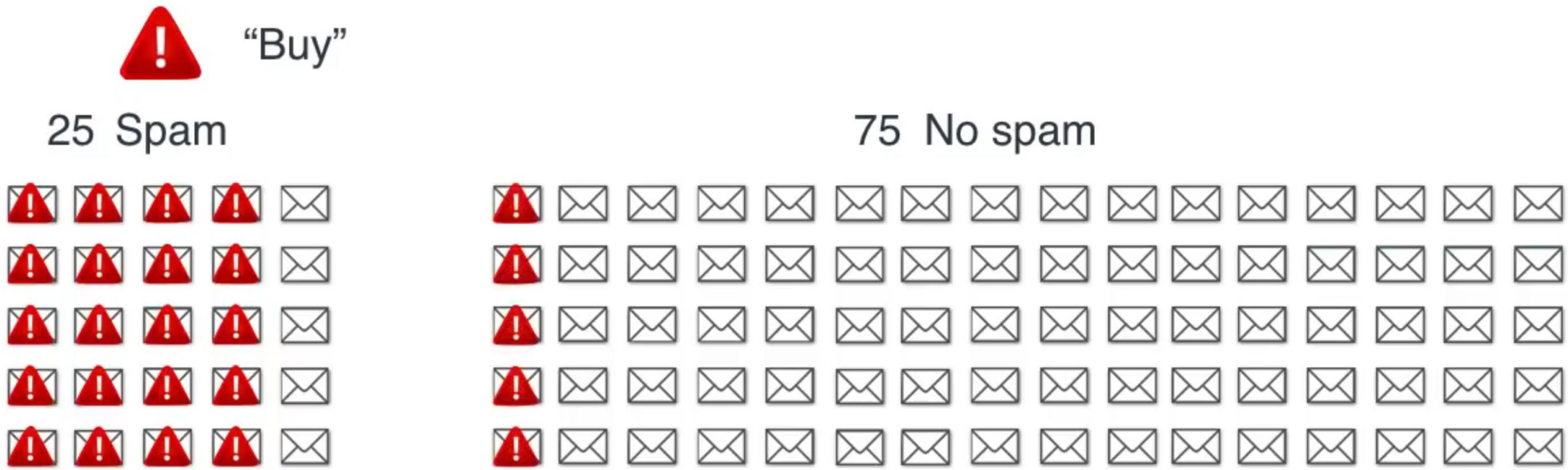
Start with an Example



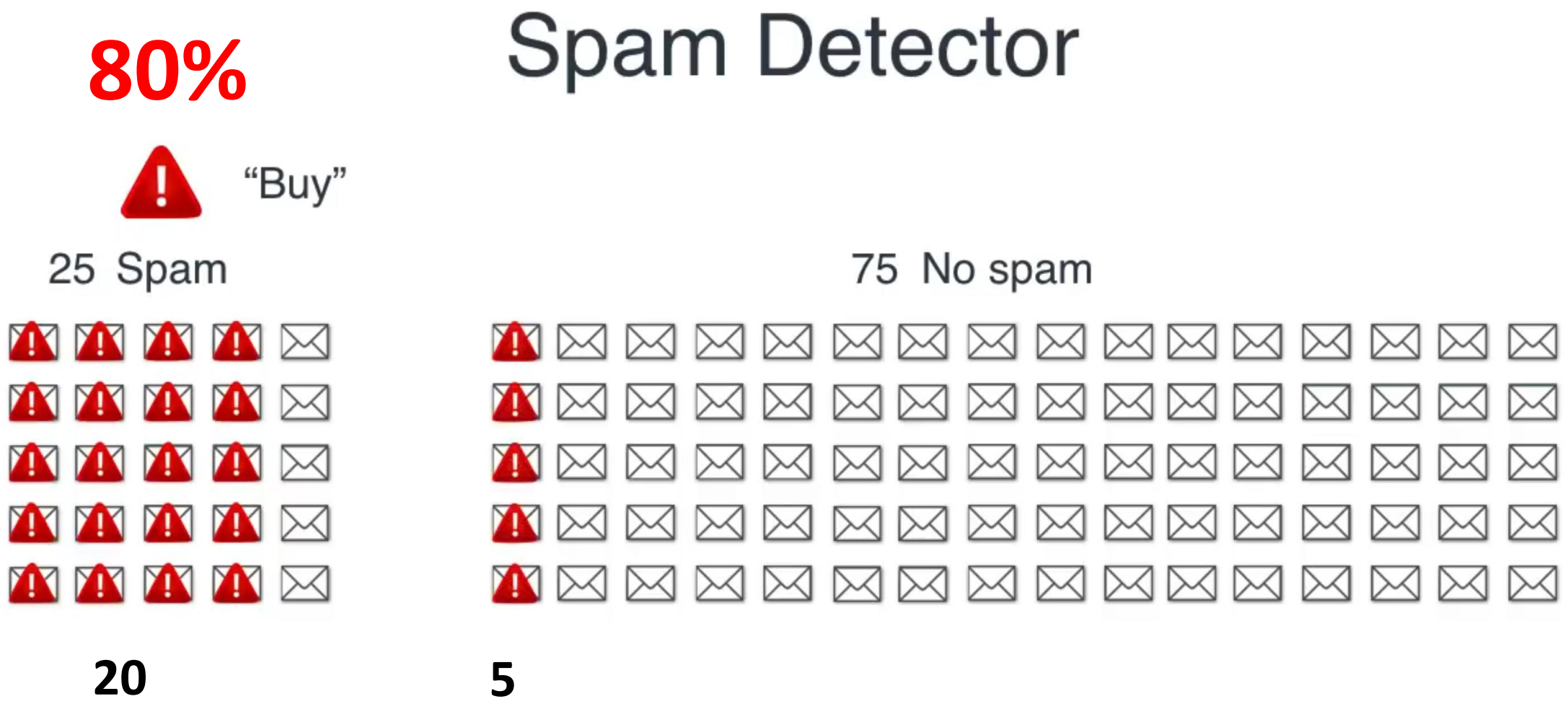
Start with an Example



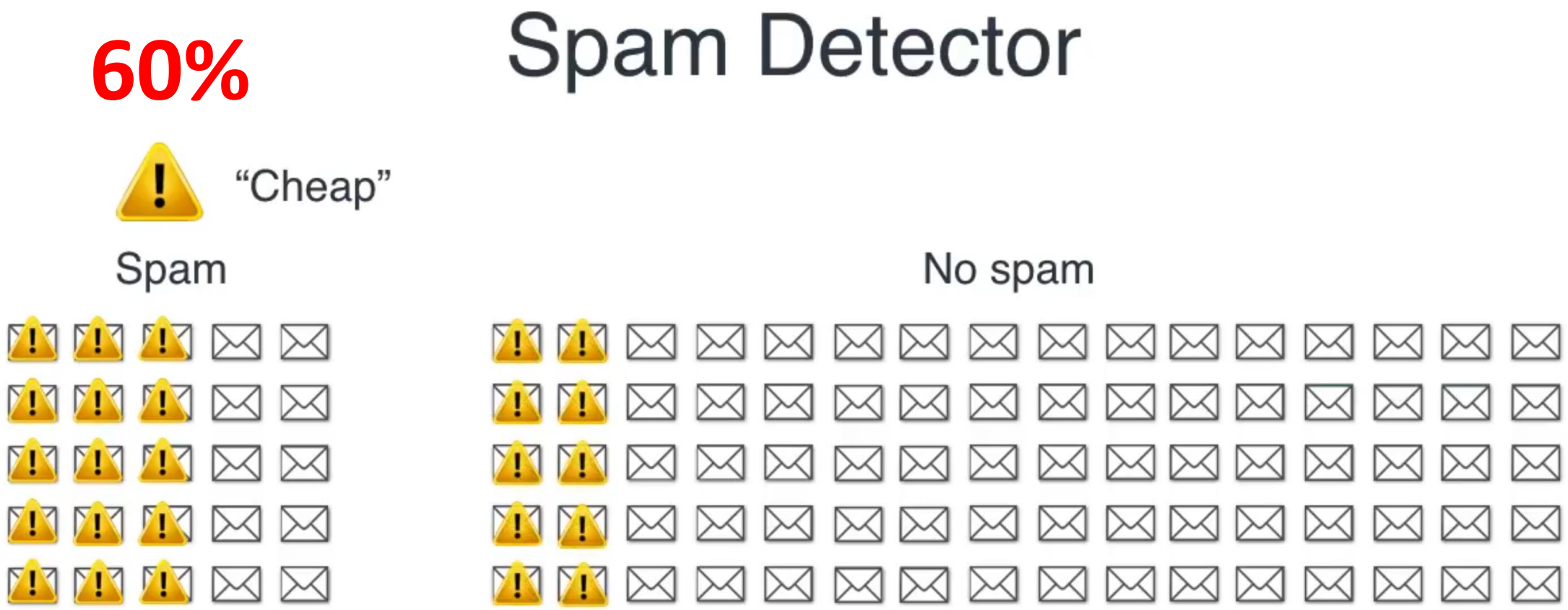
Spam Detector



If an email contains “Buy” it has an 80% chance of being spam!

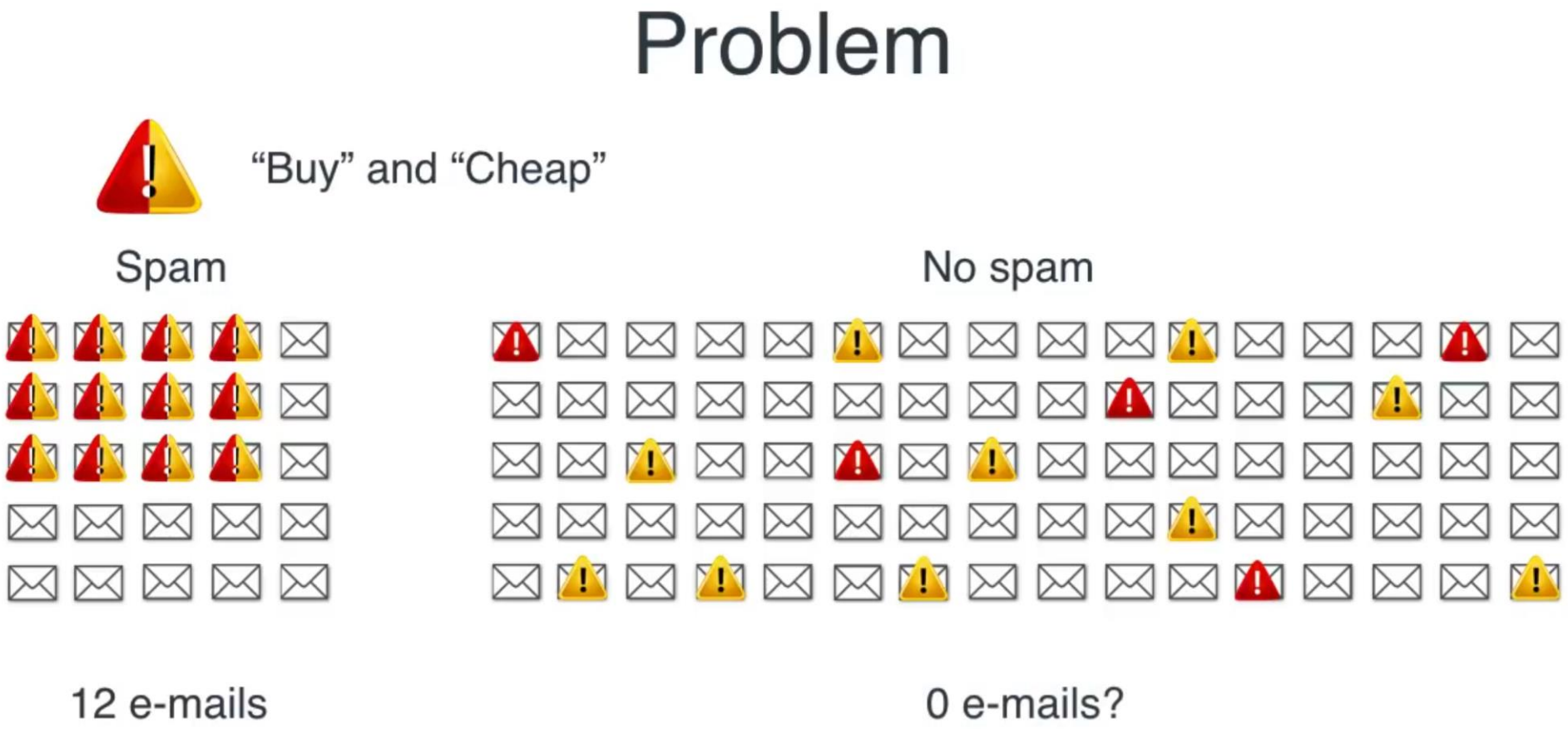


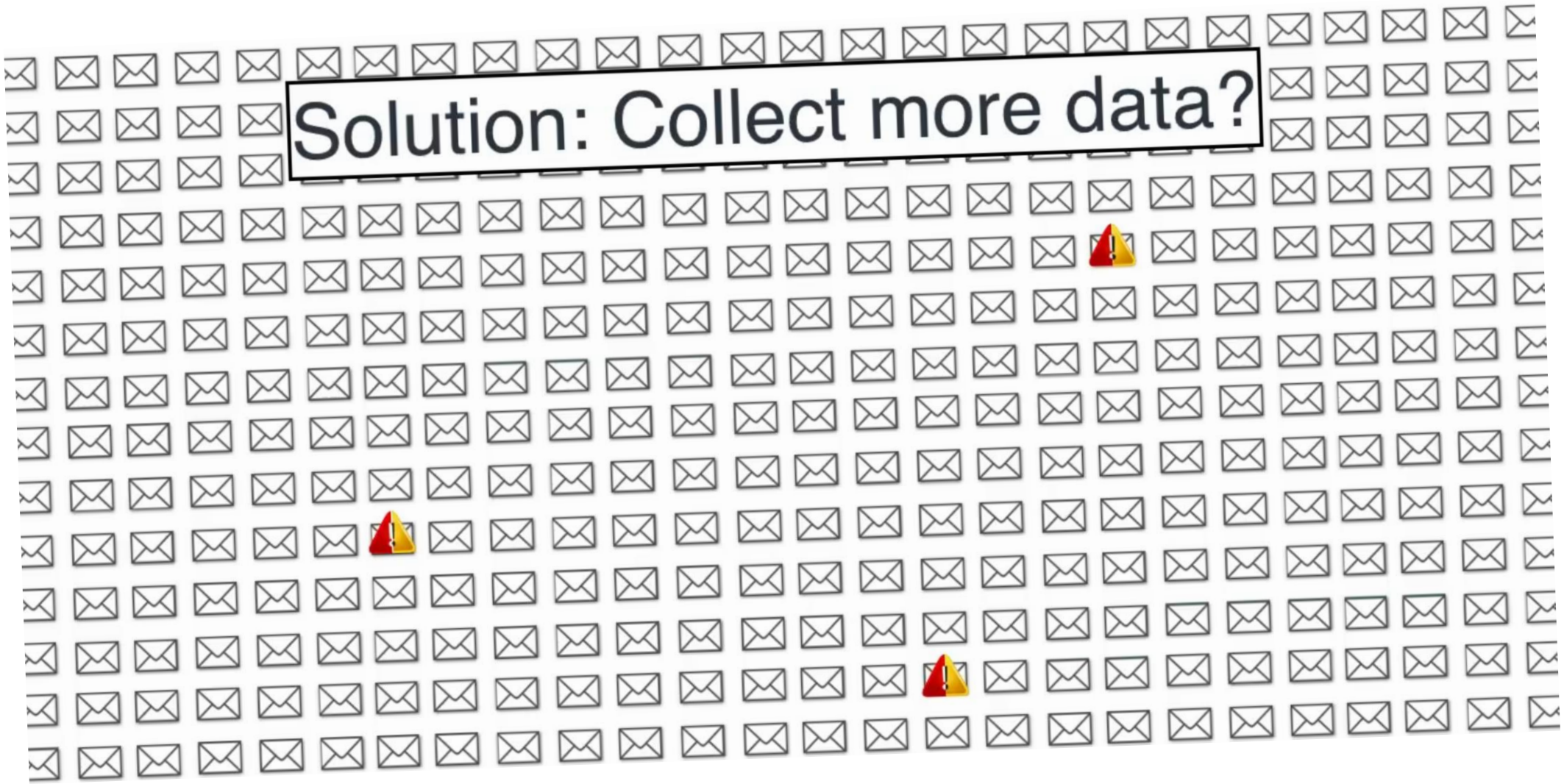
If an email contains “Cheap” it has a 60% chance of being spam!



Now we are interested in ”Buy” and “Cheap” together

If an email has “Buy & Cheap” is 100% Spam?




Solution: Collect more data?

Spam Detector

"Buy" and "Cheap"

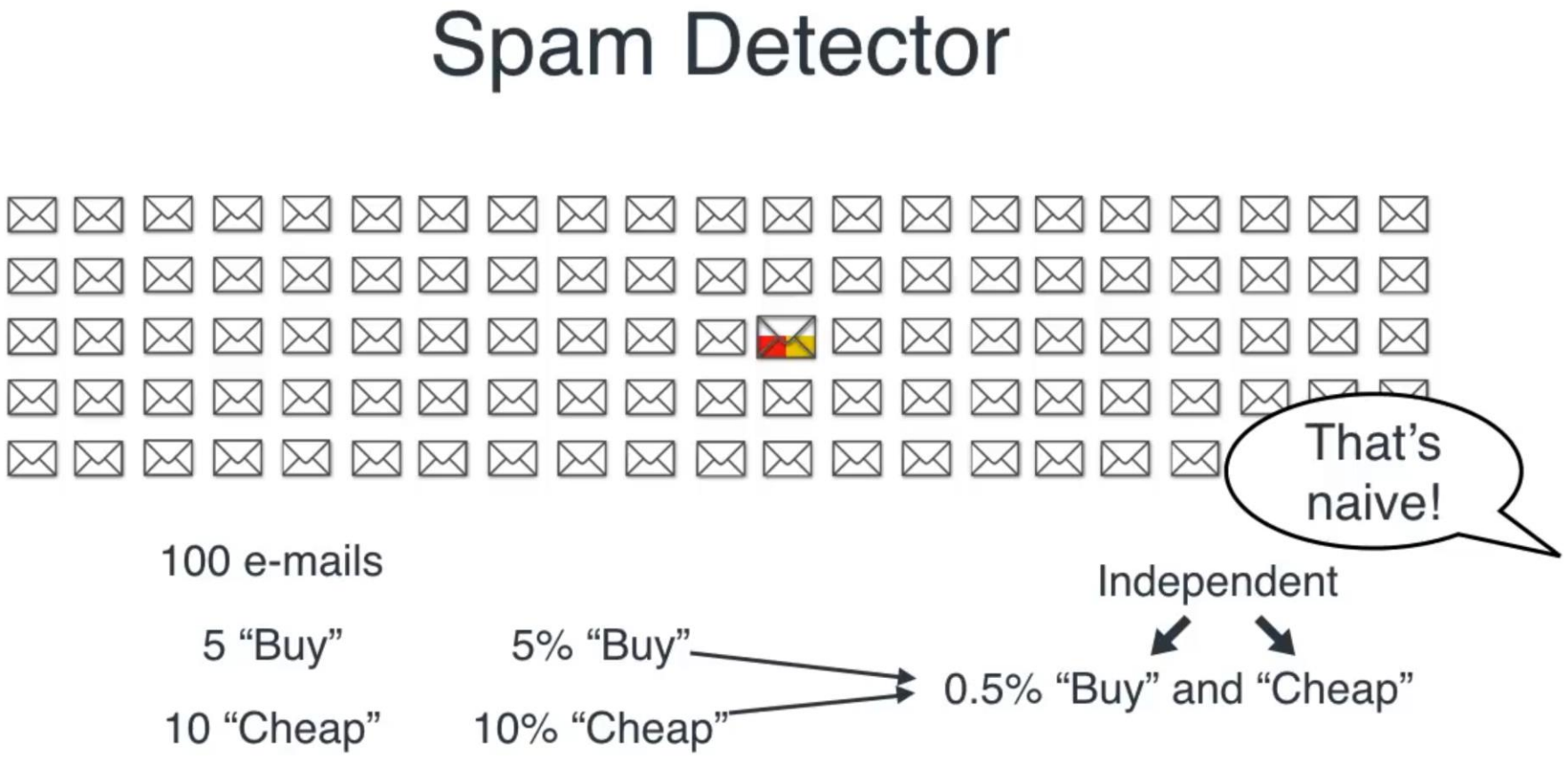
Spam

12 e-mails

No spam

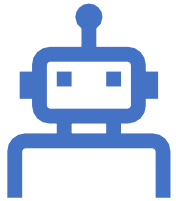
0 e-mails?

Naïve Assumption



Note: Independent VS Mutually Exclusive

- Flipping a coin and rolling a dice at the same time.
  - Probability of 4 on dice and Tail in coin?
    - Independent: 1/6 * 1/2 = 1/12



- Probability of Snowing in a 50 °C day?
  - Both cannot happen!!!
    - Mutually Exclusive

Spam Detector

Spam

25 e-mails
20 “Buy”
15 Cheap

4/5
3/5 → 12/25 × 25 = 12 “Buy” and “Cheap”

Spam Detector

No spam

75 e-mails

5 “Buy”

10 “Cheap”

1/15

2/15

2/225 × 75 = 2/3 “Buy” and “Cheap”

Spam Detector

"Buy" and "Cheap"

Spam

12

No spam

2/3

Naive Bayes Classifier

"Buy" and "Cheap" ⟶ 94.737%

Spam | No spam

 12 |  2/3

94.737% | 5.263%

$$\frac{12}{12 + 2/3} = \frac{36}{38}$$

$$= 94.737\%$$

Naive Bayes

	Spam		No spam	
Total	25		75	
Buy				
Cheap				
Buy & Cheap				

Naive Bayes

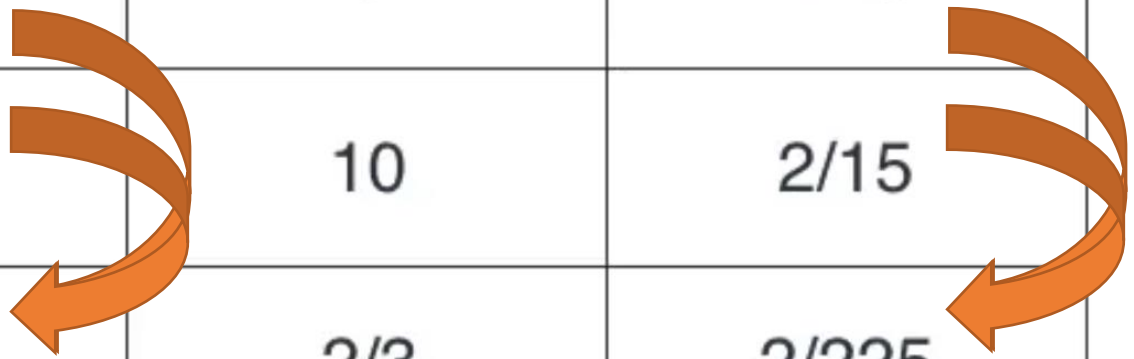
	Spam		No spam	
Total	25		75	
Buy	20	4/5	5	1/15
Cheap				
Buy & Cheap				

Naive Bayes

	Spam		No spam	
Total	25		75	
Buy	20	4/5	5	1/15
Cheap	15	3/5	10	2/15
Buy & Cheap				

Naive Bayes

	Spam		No spam	
Total	25		75	
Buy	20	4/5	5	1/15
Cheap	15	3/5	10	2/15
Buy & Cheap	12	12/25	2/3	2/225



Naive Bayes

	Spam		No spam	
Total	25		75	
Buy	20	4/5	5	1/15
Cheap	15	3/5	10	2/15
Buy & Cheap	12	12/25	2/3	2/225

$$\frac{12}{12 + 2/3} = \frac{36}{38} = 94.737\%$$

Naive Bayes

	Spam		No Spam	
Total	25		75	
Buy	20	4/5	5	1/15
Cheap	15	3/5	10	2/15
Work	5	1/5	30	6/15
Buy, Cheap, & Work				

Naive Bayes

	Spam		No Spam	
Total	25		75	
Buy	20	4/5	5	1/15
Cheap	15	3/5	10	2/15
Work	5	1/5	30	6/15
Buy, Cheap, & Work	12/5	12/125	4/15	12/3375

$$\frac{12/5}{12/5 + 4/15} = \frac{36}{40} = 90\%$$

	Spam		No spam	
Total	25		75	
Buy	20	4/5	5	1/15

Bayes Theorem

S: Spam
H: Ham (not spam)
B: 'Buy'

$$P(S \mid B) = \frac{P(B \mid S)\,P(S)}{P(B \mid S)\,P(S) + P(B \mid H)\,P(H)}$$

$$P(\text{spam if "Buy"}) = \frac{\frac{20}{25}\ \frac{25}{100}}{\frac{20}{25}\ \frac{25}{100} + \frac{5}{75}\ \frac{75}{100}} = 80\%$$

Naive Bayes

P("Buy" & "Cheap") = P("Buy") P("Cheap")

$$P(B \cap C) = P(B)\,P(C)$$

Naive Bayes

S: Spam
H: Ham (not spam)
B: ‘Buy’
C: ‘Cheap’

	Spam		No spam	
Total	25		75	
Buy	20	4/5	5	1/15
Cheap	15	3/5	10	2/15
Buy & Cheap	12	12/25	2/3	2/225

$$P(S \mid B \cap C) = \frac{P(B|S)\,P(C|S)\,P(S)}{P(B|S)\,P(C|S)\,P(S) + P(B|H)\,P(C|H)\,P(H)}$$

$$P(\text{spam if “Buy” \& “Cheap”}) = \frac{\frac{20}{25}\,\frac{15}{25}\,\frac{25}{100}}{\frac{20}{25}\,\frac{15}{25}\,\frac{25}{100} + \frac{5}{75}\,\frac{10}{75}\,\frac{75}{100}}$$

$$= 94.737\%$$

The impact of viral diseases such as COVID-19 is very much top of mind nowadays.

Let's assume 0.1% of Earth's population is infected by a particular virus.

A randomly selected person tests positive for the virus, and asks their doctor "what is the accuracy of these tests?" The doctor replies "This test correctly identifies 99% of the people who have the virus, and incorrectly identifies 1% of the people who don't have the virus."

Calculate the probability of a person being infected with the virus, if the result of this test is positive?

The impact of viral diseases such as COVID-19 is very much top of mind nowadays.

Let's assume 0.1% of Earth's population is infected by a particular virus.

A randomly selected person tests positive for the virus, and asks their doctor "what is the accuracy of these tests?" The doctor replies "This test correctly identifies 99% of the people who have the virus, and incorrectly identifies 1% of the people who don't have the virus."

Calculate the probability of a person being infected with the virus, if the result of this test is positive?

We are told:

P(Virus = Yes)	0.001
P(Test = Positive | Virus = Yes)	0.99
P(Test = Positive | Virus = No)	0.01

We want:

P(Virus = Yes | Test = Positive)	

		P(Test = Positive)		P(Test = Negative)	
P(Virus = Yes)					
P(Virus = No)					

		P(Test = Positive)		P(Test = Negative)	
P(Virus = Yes)		P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)		P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)		P(Virus = No)	
P(Test = Positive)		P(Test = Positive | Virus = Yes)		P(Test = Positive | Virus = No)	
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

		P(Test = Positive)		P(Test = Negative)	
P(Virus = Yes)		P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)		P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)		P(Virus = No)	
P(Test = Positive)		P(Test = Positive | Virus = Yes)		P(Test = Positive | Virus = No)	
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

P(Virus = Yes)	0.001
P(Test = Positive | Virus = Yes)	0.99
P(Test = Positive | Virus = No)	0.01

		P(Test = Positive)		P(Test = Negative)	
P(Virus = Yes)	0.001	P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)		P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)	0.001	P(Virus = No)	
P(Test = Positive)		P(Test = Positive | Virus = Yes)	0.990	P(Test = Positive | Virus = No)	0.010
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

		P(Test = Positive)		P(Test = Negative)	
P(Virus = Yes)	0.001	P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)		P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)	0.001	P(Virus = No)	
P(Test = Positive)		P(Test = Positive | Virus = Yes)	0.990	P(Test = Positive | Virus = No)	0.010
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

		P(Test = Positive)		P(Test = Negative)	
P(Virus = Yes)	0.001	P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)	0.999	P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)	0.001	P(Virus = No)	0.999
P(Test = Positive)		P(Test = Positive | Virus = Yes)	0.990	P(Test = Positive | Virus = No)	0.010
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

		P(Test = Positive)		P(Test = Negative)	
P(Virus = Yes)	0.001	P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)	0.999	P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)	0.001	P(Virus = No)	0.999
P(Test = Positive)		P(Test = Positive | Virus = Yes)	0.990	P(Test = Positive | Virus = No)	0.010
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

		P(Test = Positive)	0.01098	P(Test = Negative)	
P(Virus = Yes)	0.001	P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)	0.999	P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)	0.001	P(Virus = No)	0.999
P(Test = Positive)	0.01098	P(Test = Positive | Virus = Yes)	0.990	P(Test = Positive | Virus = No)	0.010
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

P(Test = Positive) = P(Test = Positive | Virus = Yes) * P(Virus = Yes) + P(Test = Positive | Virus = No)*P(Virus = No)

		P(Test = Positive)	0.01098	P(Test = Negative)	
P(Virus = Yes)	0.001	P(Virus = Yes | Test = Positive)		P(Virus = Yes | Test = Negative)	
P(Virus = No)	0.999	P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)	0.001	P(Virus = No)	0.999
P(Test = Positive)	0.01098	P(Test = Positive | Virus = Yes)	0.990	P(Test = Positive | Virus = No)	0.010
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

P(Test = Positive) = P(Test = Positive | Virus = Yes) * P(Virus = Yes) + P(Test = Positive | Virus = No)*P(Virus = No)

		P(Test = Positive)	0.01098	P(Test = Negative)	
P(Virus = Yes)	0.001	P(Virus = Yes | Test = Positive)	0.090	P(Virus = Yes | Test = Negative)	
P(Virus = No)	0.999	P(Virus = No | Test = Positive)		P(Virus = No | Test = Negative)	
		P(Virus = Yes)	0.001	P(Virus = No)	0.999
P(Test = Positive)	0.01098	P(Test = Positive | Virus = Yes)	0.990	P(Test = Positive | Virus = No)	0.010
P(Test = Negative)		P(Test = Negative | Virus = Yes)		P(Test = Negative | Virus = No)	

Bayes Theorem -> posterior = prior * likelihood / evidence

P(Virus = Yes | Test = Positive) = P(Test = Positive | Virus = Yes) * P(Virus = Yes) / P(Test = Positive)

9% seems low! – Base rate fallacy

https://machinelearningmastery.com/bayes-theorem-for-machine-learning/

The impact of viral diseases such as COVID-19 is very much top of mind nowadays.

Let's assume 0.1% of Earth's population is infected by a particular virus.

A randomly selected person tests positive for the virus, and asks their doctor "what is the accuracy of these tests?" The doctor replies "This test correctly identifies 99% of the people who have the virus, and incorrectly identifies 1% of the people who don't have the virus."

Calculate the probability of a person being infected with the virus, if the result of this test is positive?

$$P(Test = Positive \mid Virus = Yes) = 0.99$$

$$P(Virus = Yes) = 0.001$$

$$P(Virus = No) = 1 - P(Virus = Yes) = 1 - 0.001 = 0.999$$

$$P(Test = Positive \mid Virus = No) = 0.01$$

$$\begin{aligned} P(Test = Positive) \\ = P(Test = Positive \mid Virus = Yes) * P(Virus = Yes) \\ + P(Test = Positive \mid Virus = No) * P(Virus = No) \end{aligned}$$

$$P(Test = Positive) = 0.99 * 0.001 + 0.01 * 0.999 = 0.01098$$

$$P(Virus = Yes \mid Test = Positive) = \frac{P(Test = Positive \mid Virus = Yes)\, P(Virus = Yes)}{P(Test = Positive)}$$

$$= \frac{0.99 * 0.001}{0.01098} = 9.0\%$$

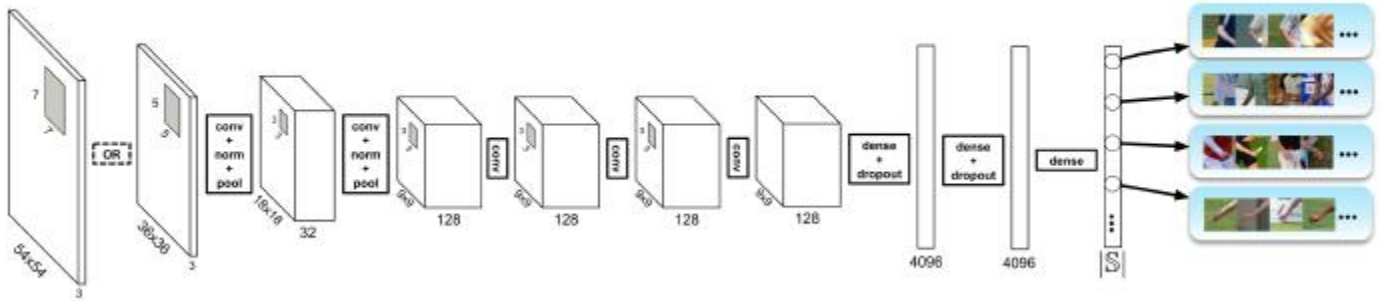
Bayesian Optimization

Machine Learning Meta-Challenges

- Machine learning models are getting more and more complicated
  - Usually more parameters (e.g., deep neural networks)
- Non-convex optimization methods have many parameters to tune
- Generally hard to apply modern techniques or reproduce results

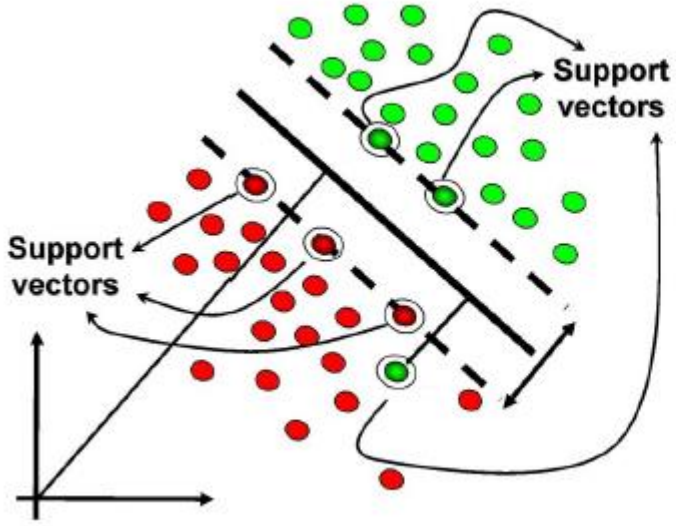
<u>Solution</u>: Automate the selection of critical hyper-parameters [see also: Automated Machine Learning (AutoML)]

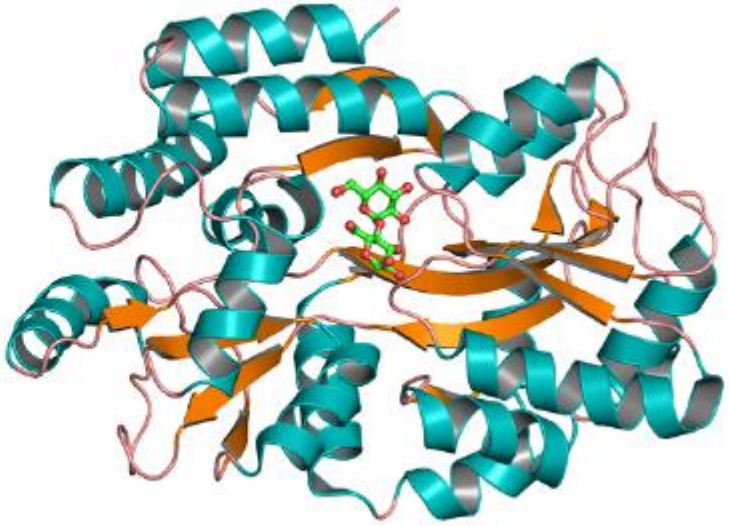
Example: Deep Neural Networks



- Huge interest in large neural networks
  - When well-tuned, very successful for visual object identification, speech recognition, computational biology, ...
- Huge investments by Google, Facebook, Microsoft, etc.
- ***Many choices:*** number of layers, weight regularization, layer size, which nonlinearity, batch size, learning rate schedule, stopping conditions

Example: Classification of DNA Sequences





- Objective: Predict which DNA sequences will bind with which proteins.
- Miller et al. (2012): Latent Structural Support Vector Machine
- ***Hyper-parameters:*** margin/slack parameter, entropy parameter, convergence criterion

Search for Good Hyper-parameters

- Define an objective function
  - Usually, we care about generalization performance.
  - Cross-validation to measure parameter quality
- Standard search procedures:
  - Grid search
  - Random search (very simple, works surprisingly well)
  - Manual tuning
- Painful:
  - Training may be very expensive (e.g., time or money)
  - Many training cycles
  - Curse of dimensionality
  - Possibly noisy

Alternative Approach: Bayesian Optimization

Setting

Globally optimize an objective function that is expensive to evaluate

(e.g., cross-validation error for a massive neural network)

- Build a ***probabilistic proxy model*** for the objective ***using outcomes of past experiments as training data***
- The proxy model is much ***cheaper to evaluate*** than the original objective
- ***Optimize cheap proxy*** function to determine where to evaluate the true objective next
- Standard proxy: ***Gaussian process***

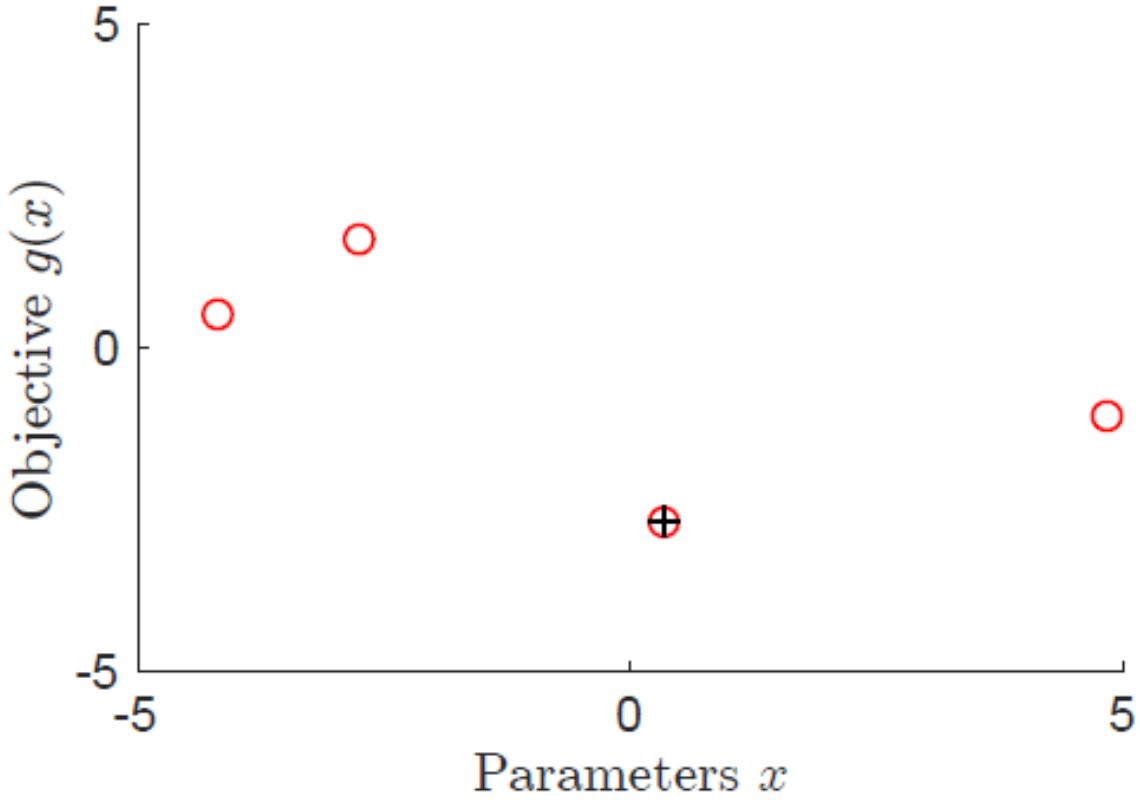
Probability distribution over possible functions,
i.e. Gaussian Mixture Model

Setting (2)

- Objective: Find global minimum of objective function *g*:

$$x_* = \arg\min_x g(x)$$

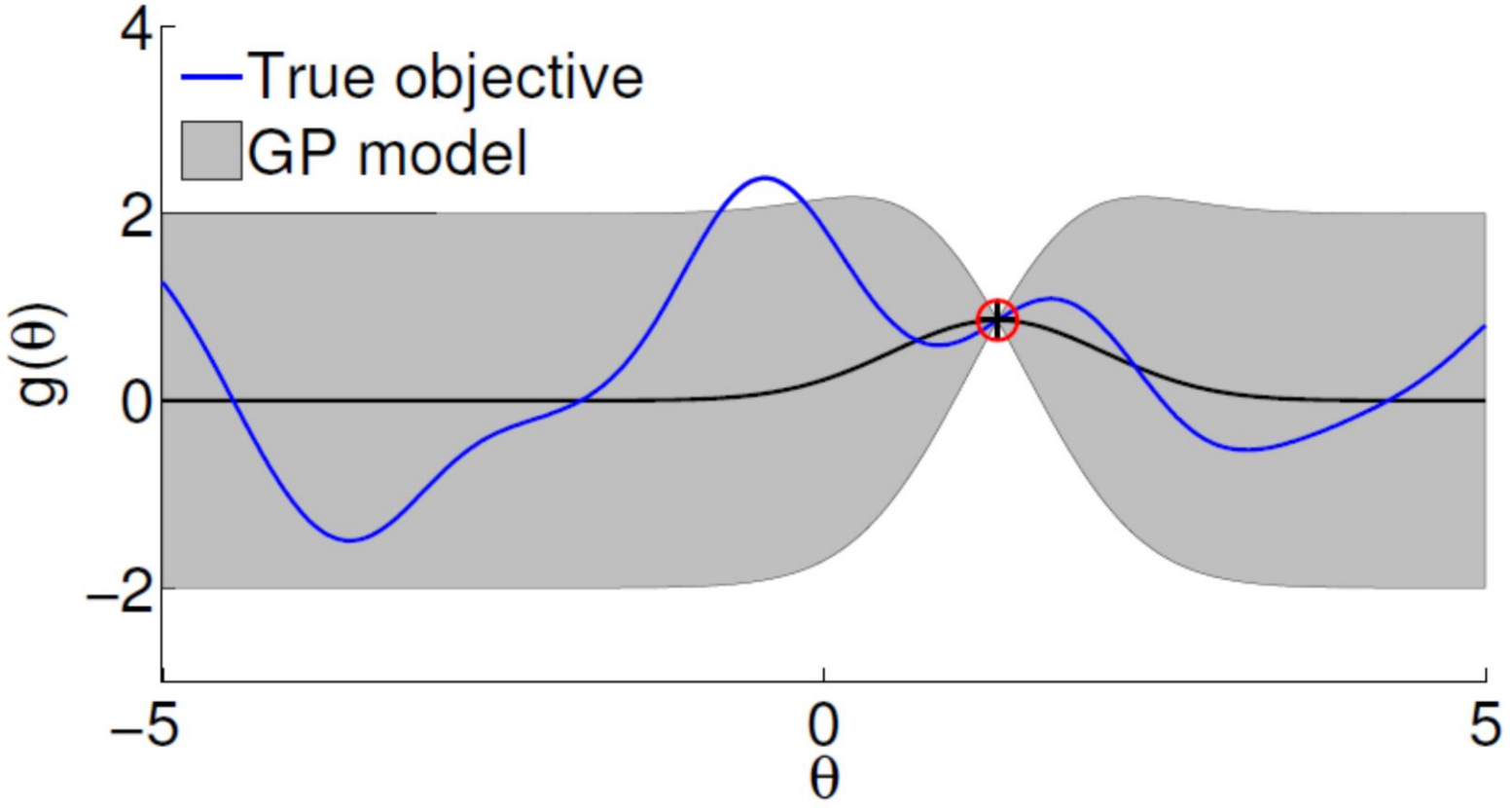
- We can evaluate the objective g pointwise, but do not have an easy functional form or gradients; observations may be noisy
- ***Evaluating g is costly*** (e.g., train a massive deep network)



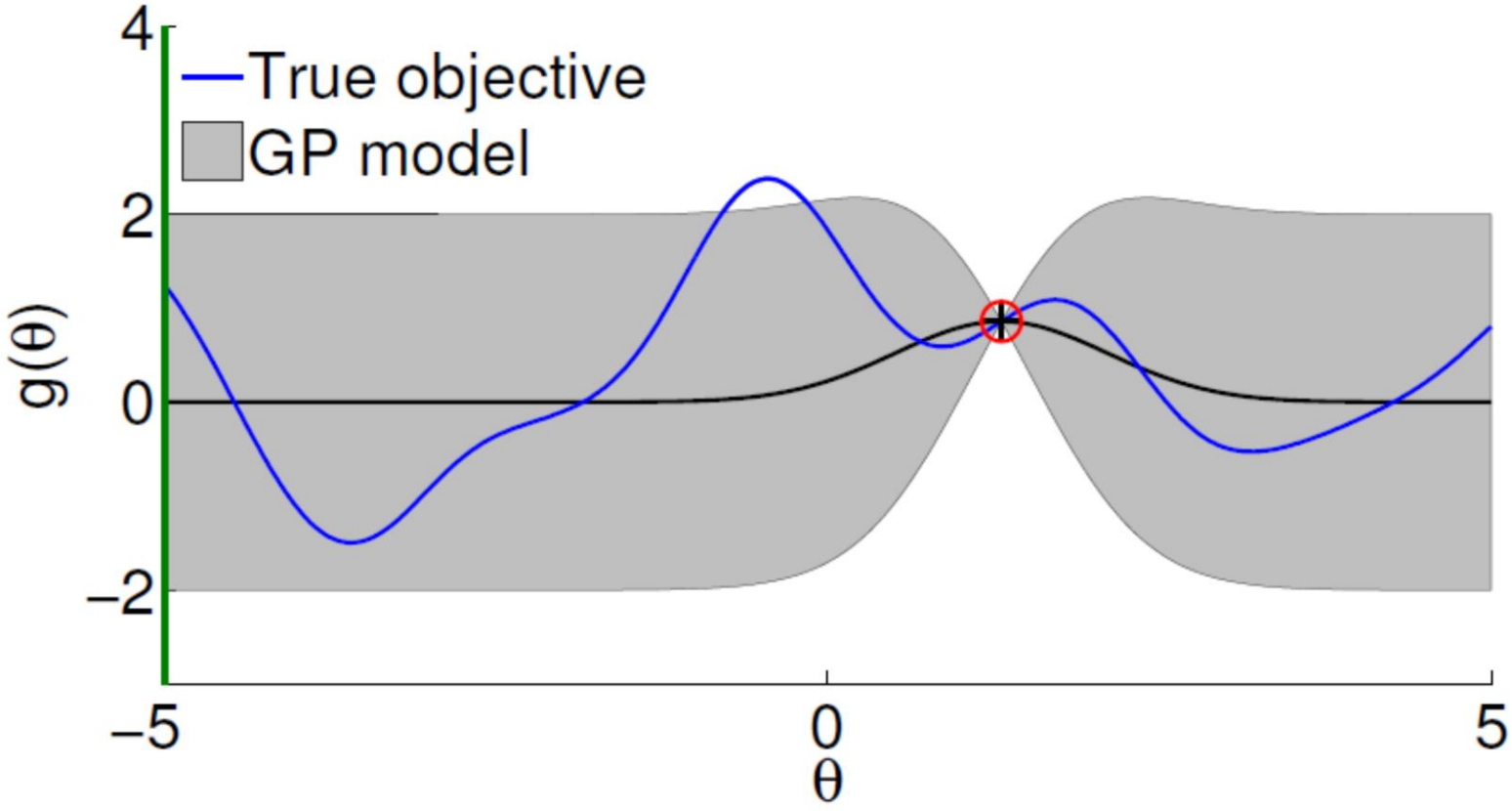
Key Steps

- To avoid evaluating *g* an excessive number of times, approximate it using a ***proxy function*** $\tilde{g}$ (which is cheap to evaluate)
- Find a ***global optimum*** $\tilde{g}(x_*)$ of ***proxy function*** $\tilde{g}$
- Evaluate true objective *g* at x_*
- Overall: Evaluate *g* only once
- Works well if $\tilde{g} \approx g$
- Usually not the case
  - Repeat this cycle and keep updating $\tilde{g}$

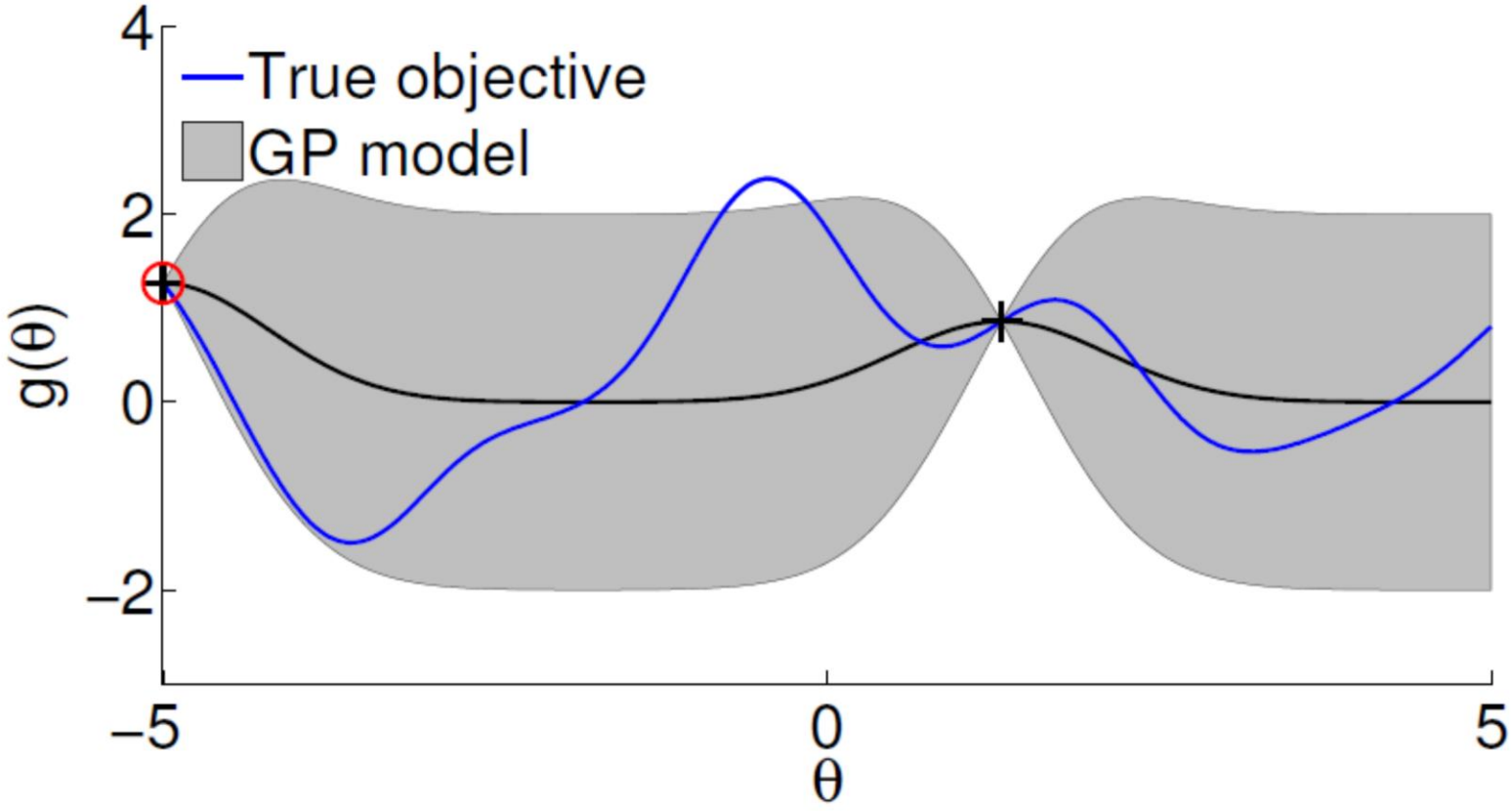
Bayesian Optimization: Illustration



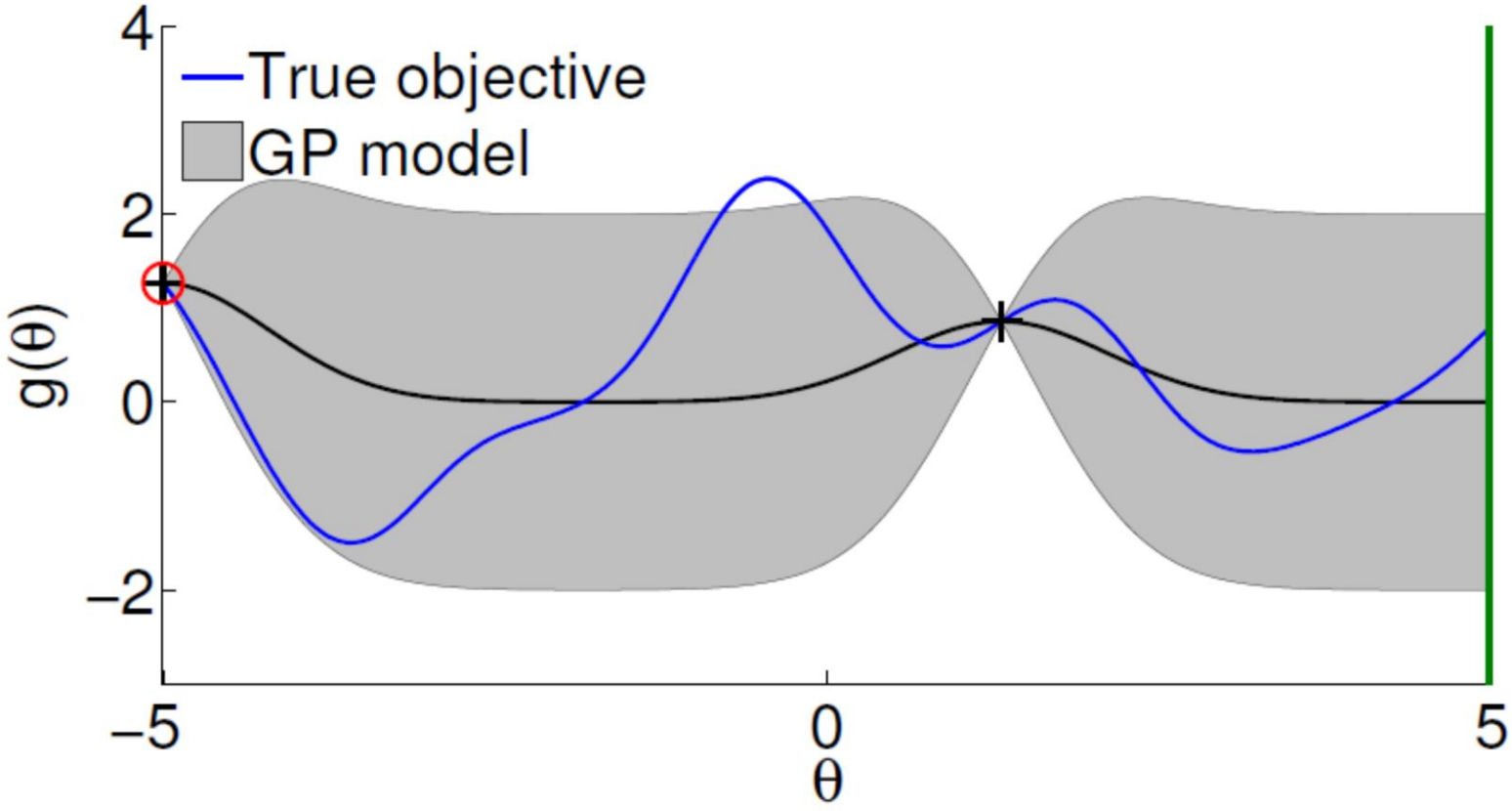
Bayesian Optimization: Illustration



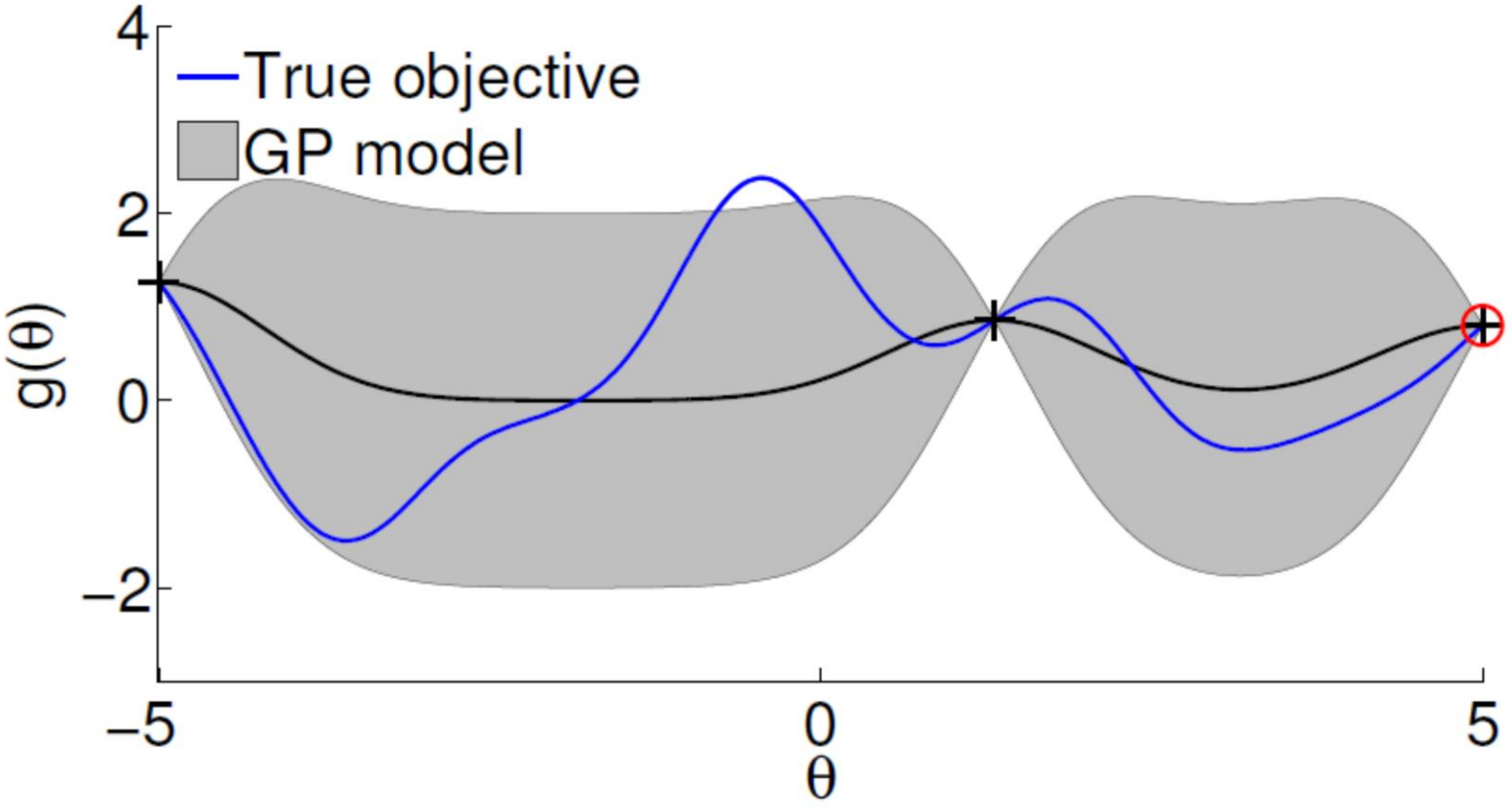
Bayesian Optimization: Illustration



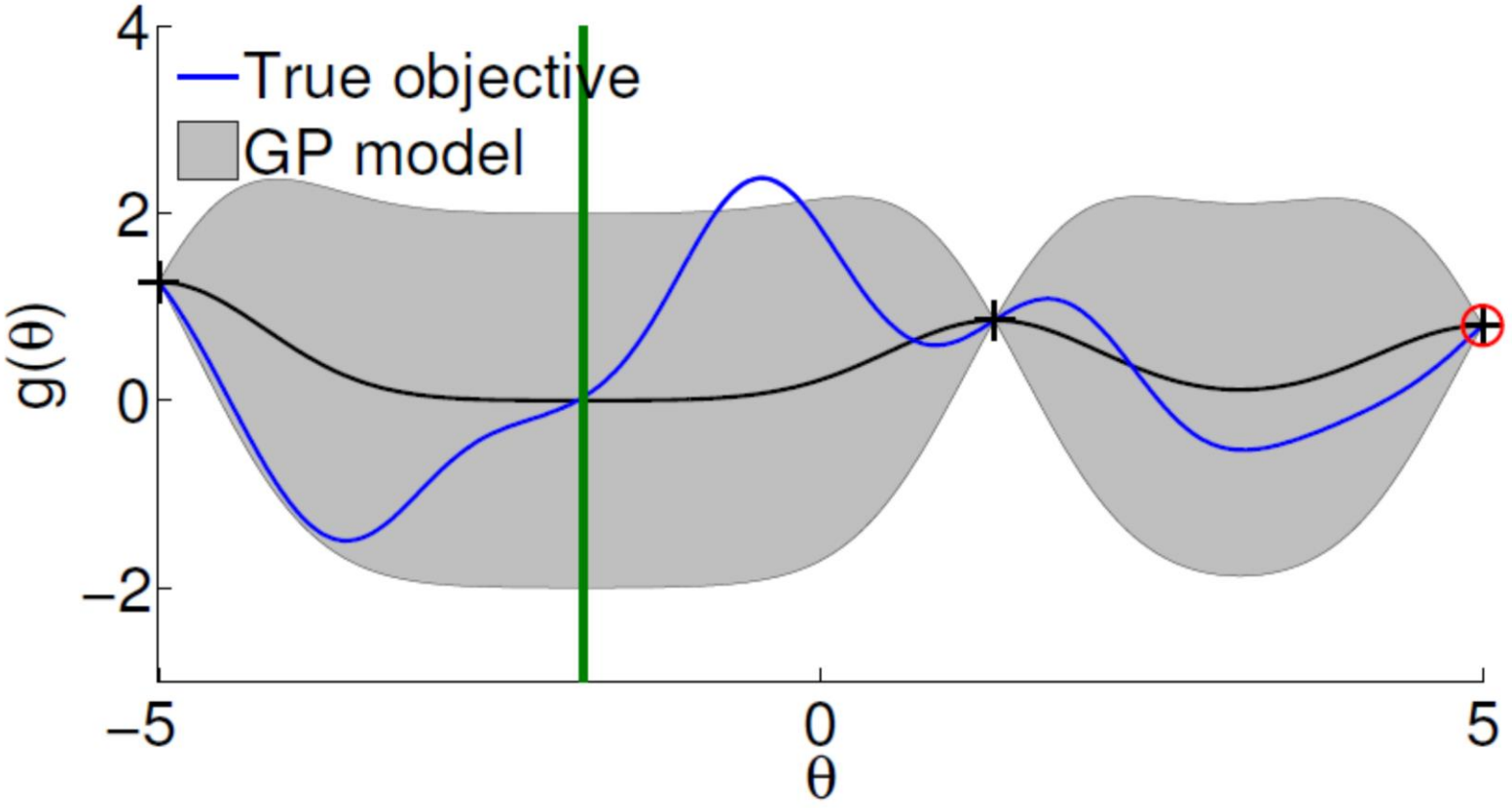
Bayesian Optimization: Illustration



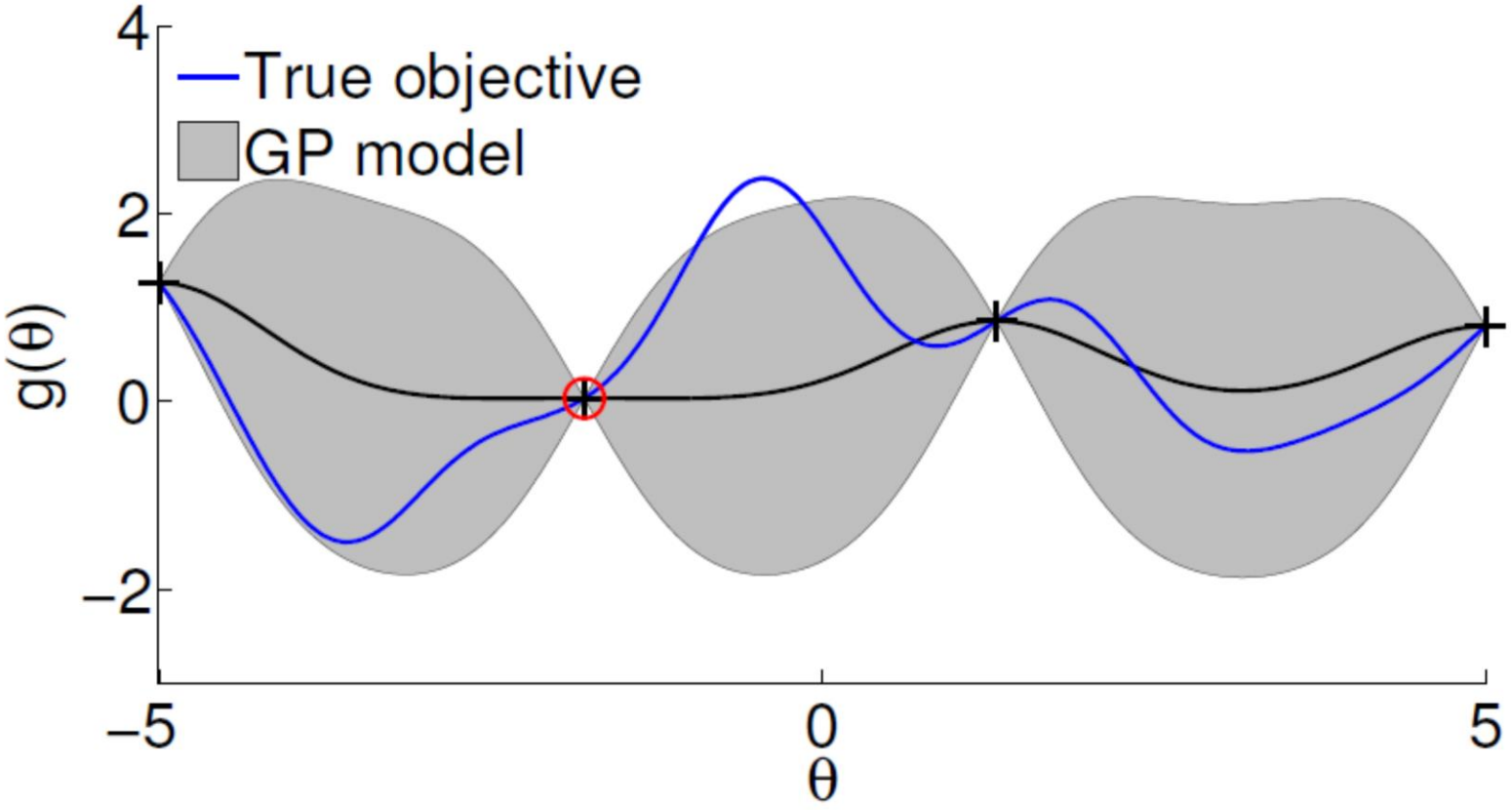
Bayesian Optimization: Illustration



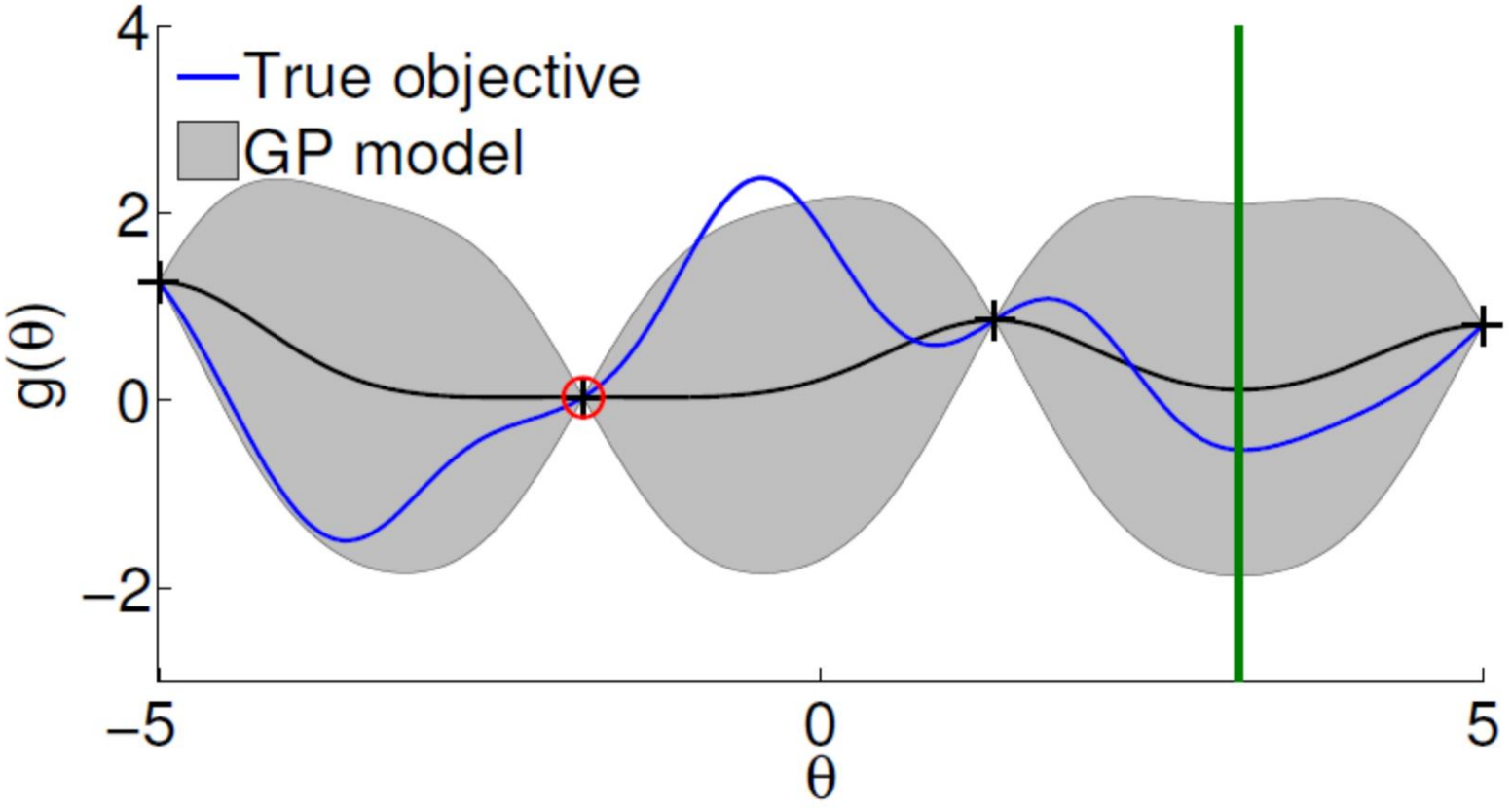
Bayesian Optimization: Illustration



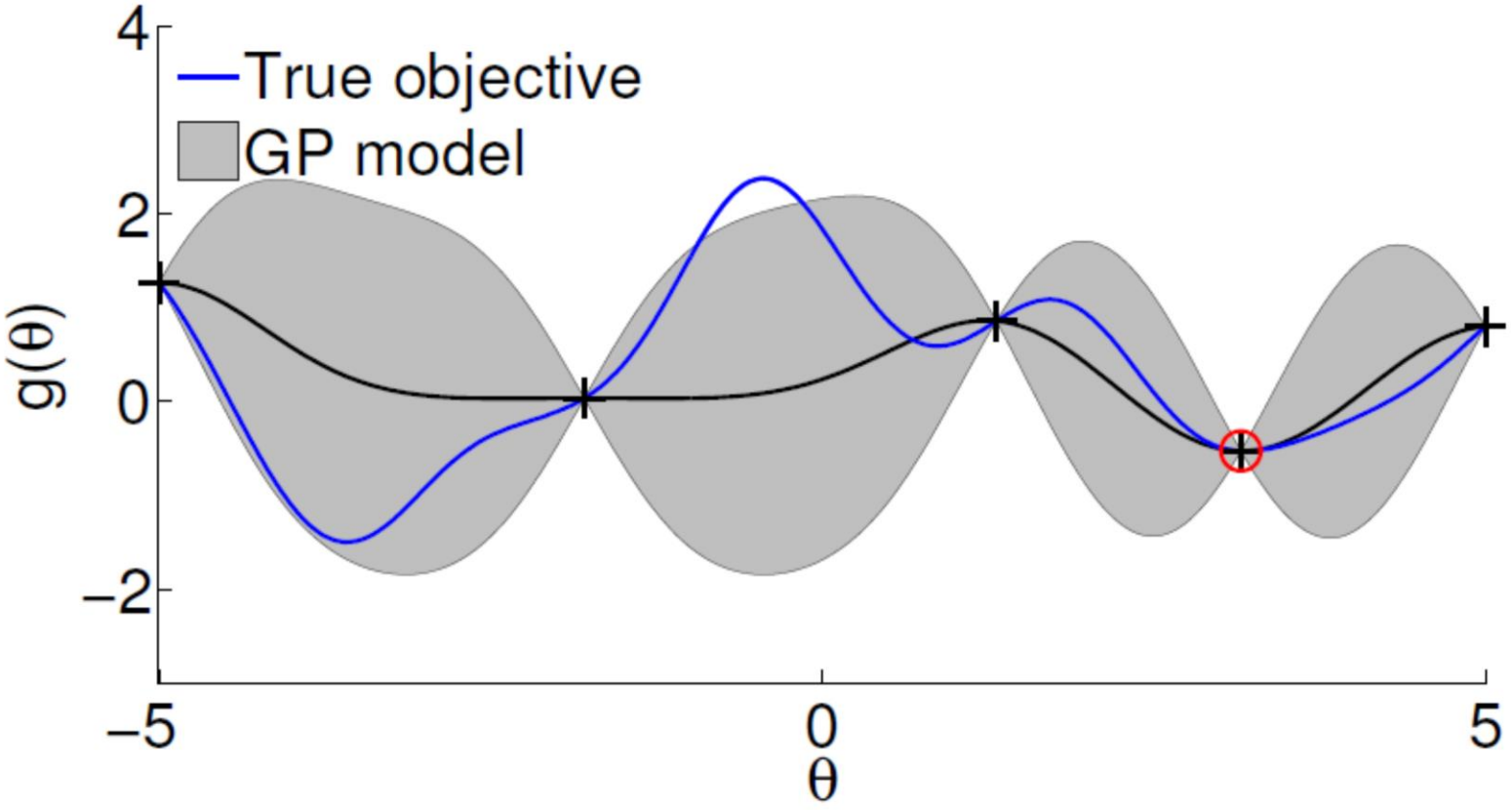
Bayesian Optimization: Illustration



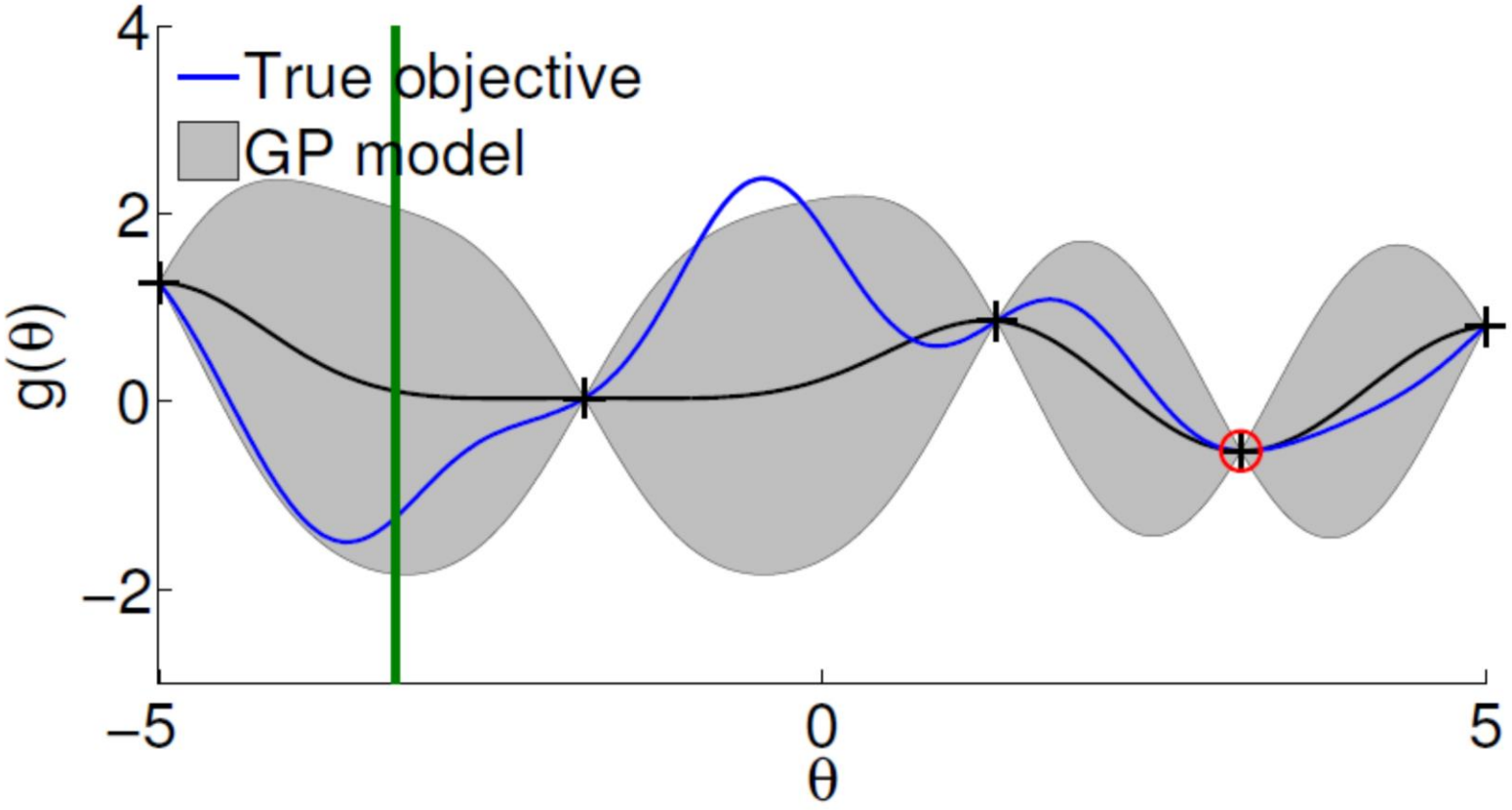
Bayesian Optimization: Illustration



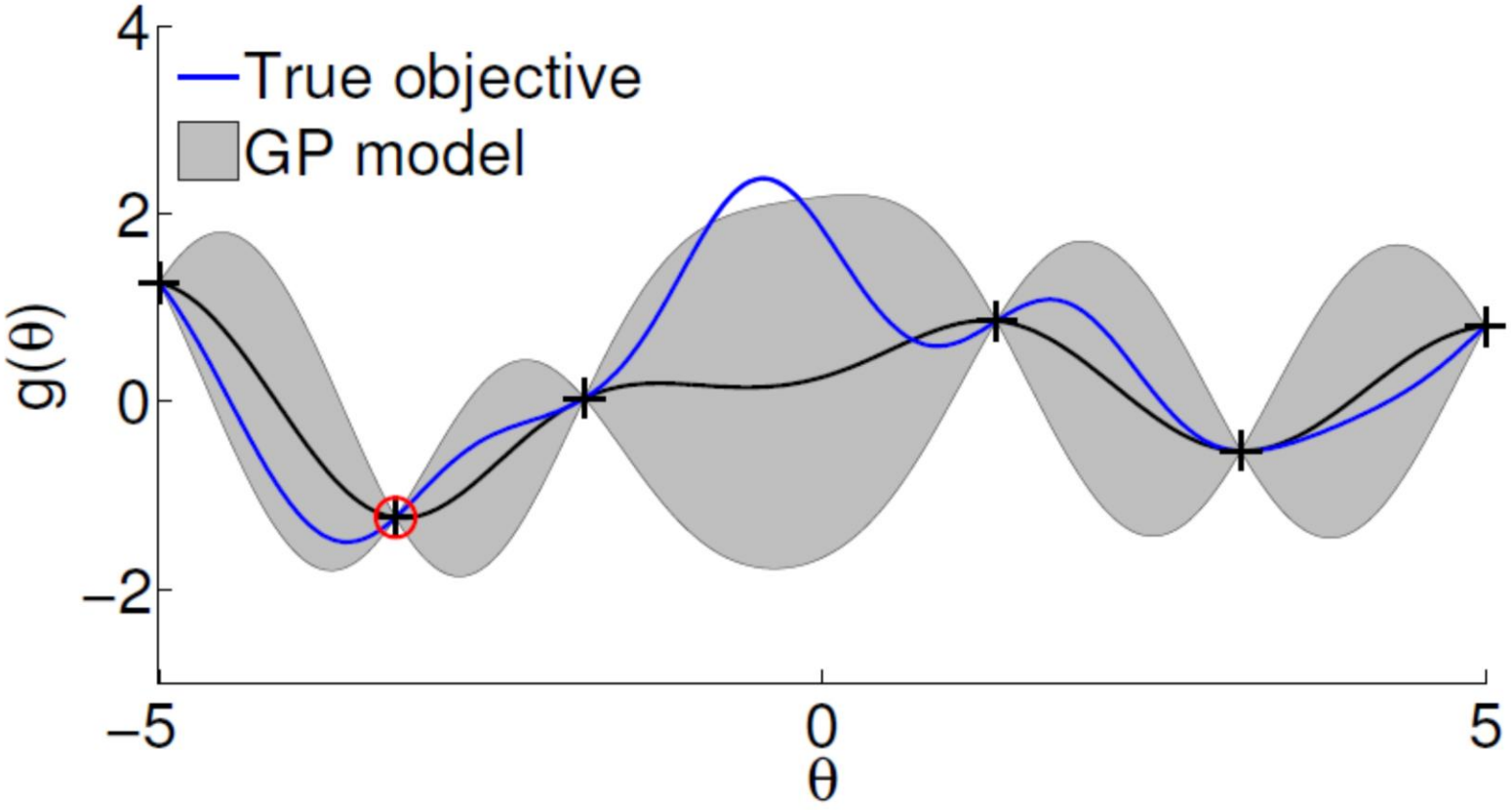
Bayesian Optimization: Illustration



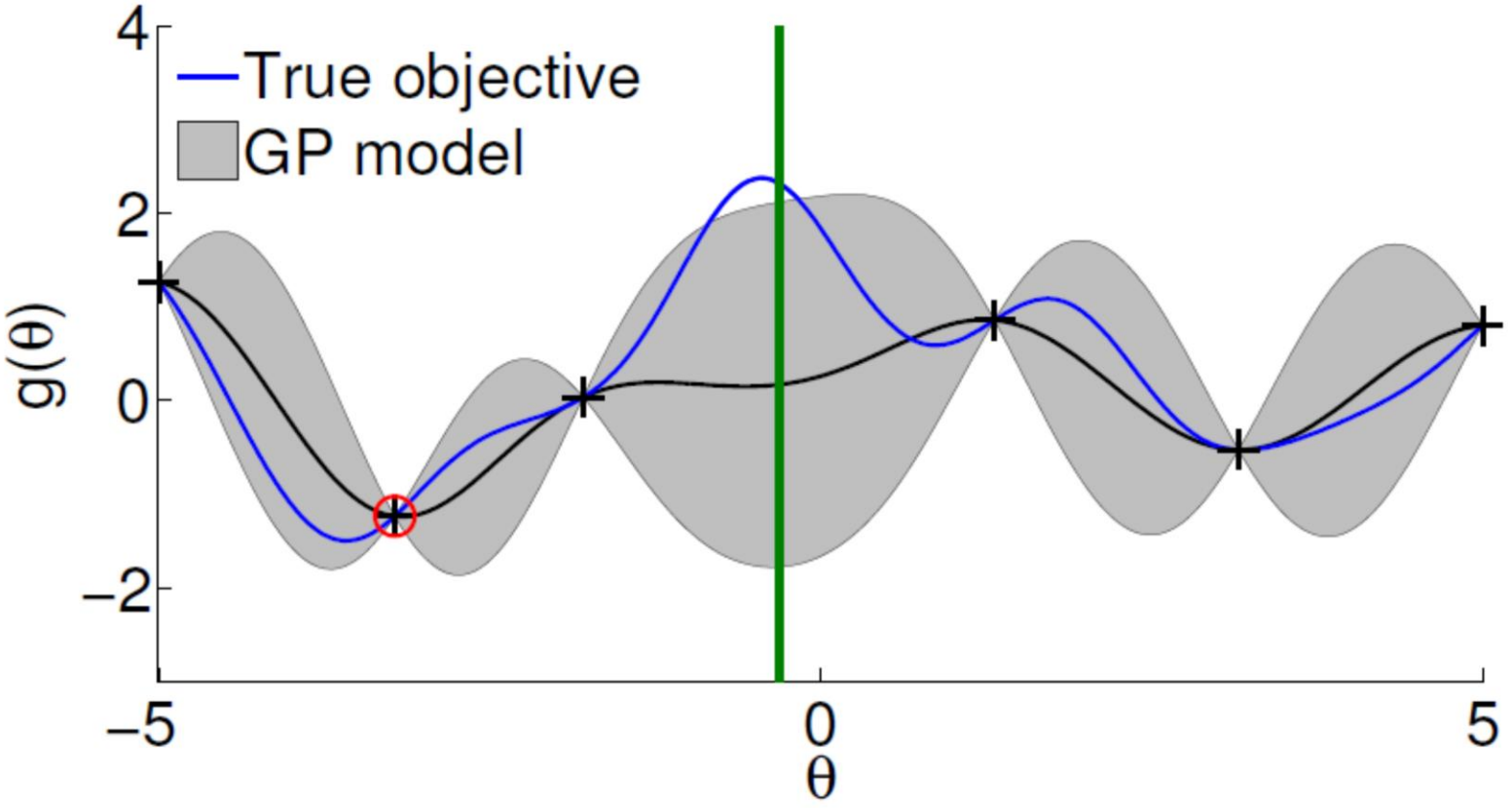
Bayesian Optimization: Illustration



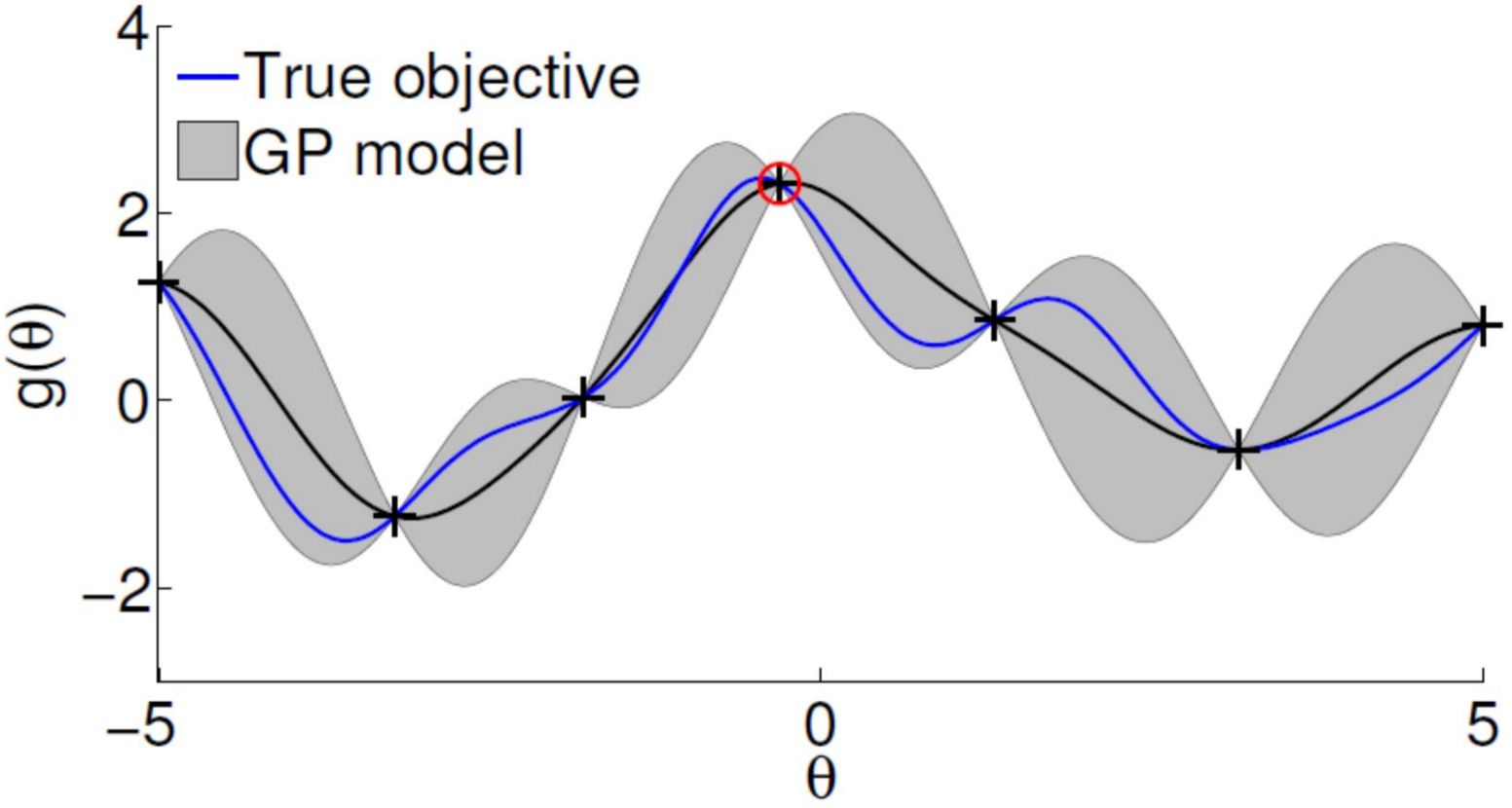
Bayesian Optimization: Illustration



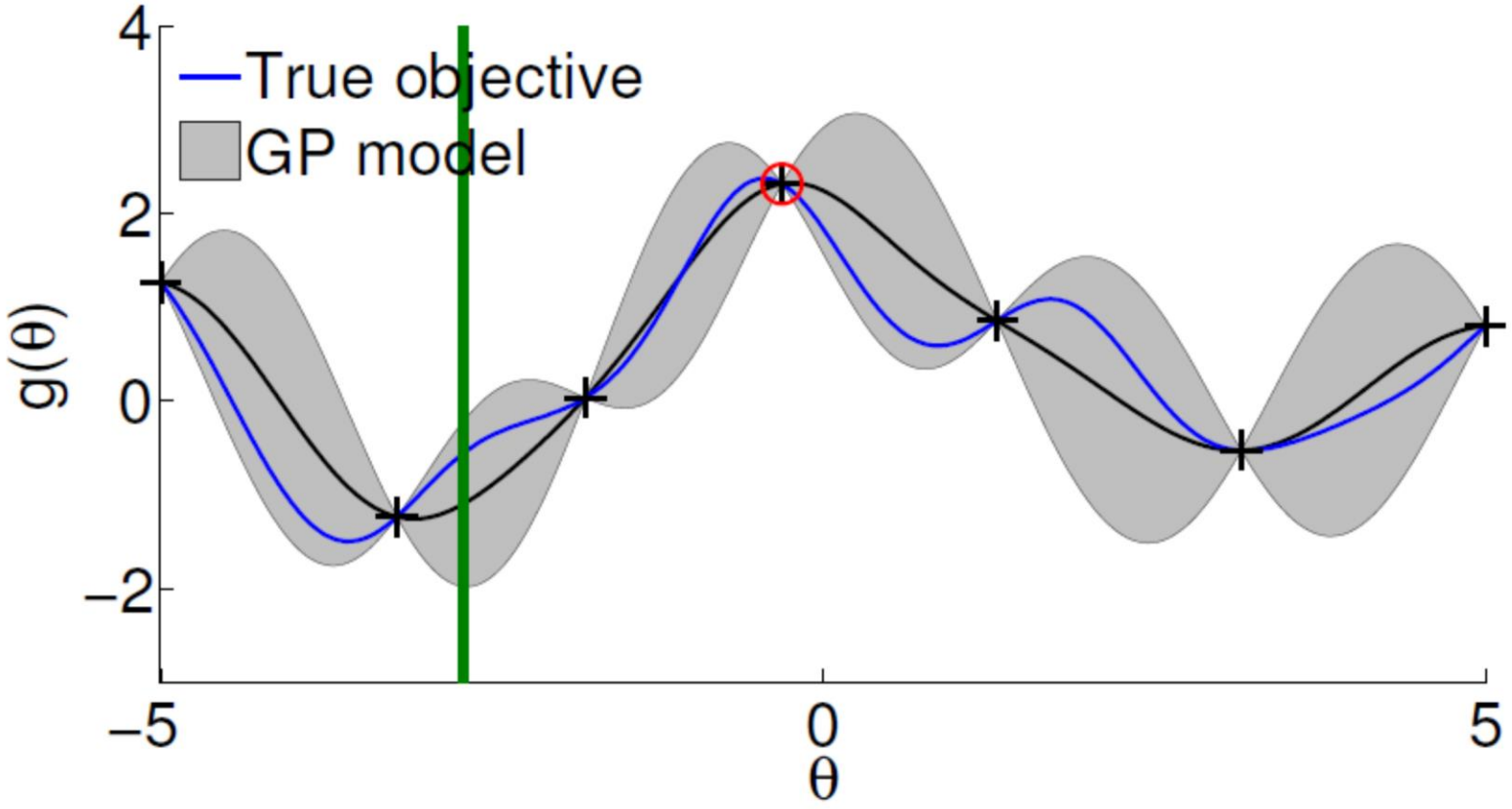
Bayesian Optimization: Illustration



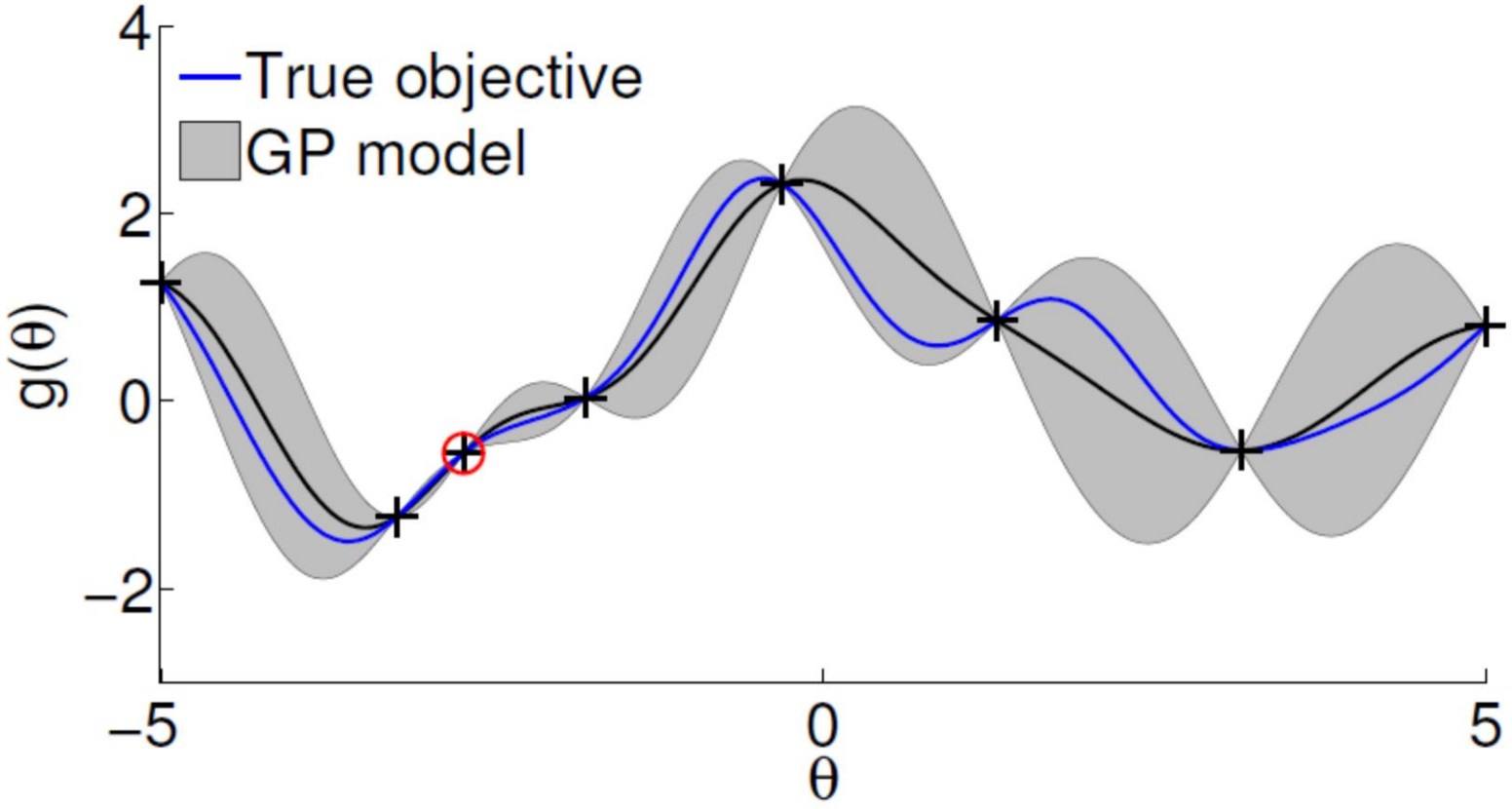
Bayesian Optimization: Illustration



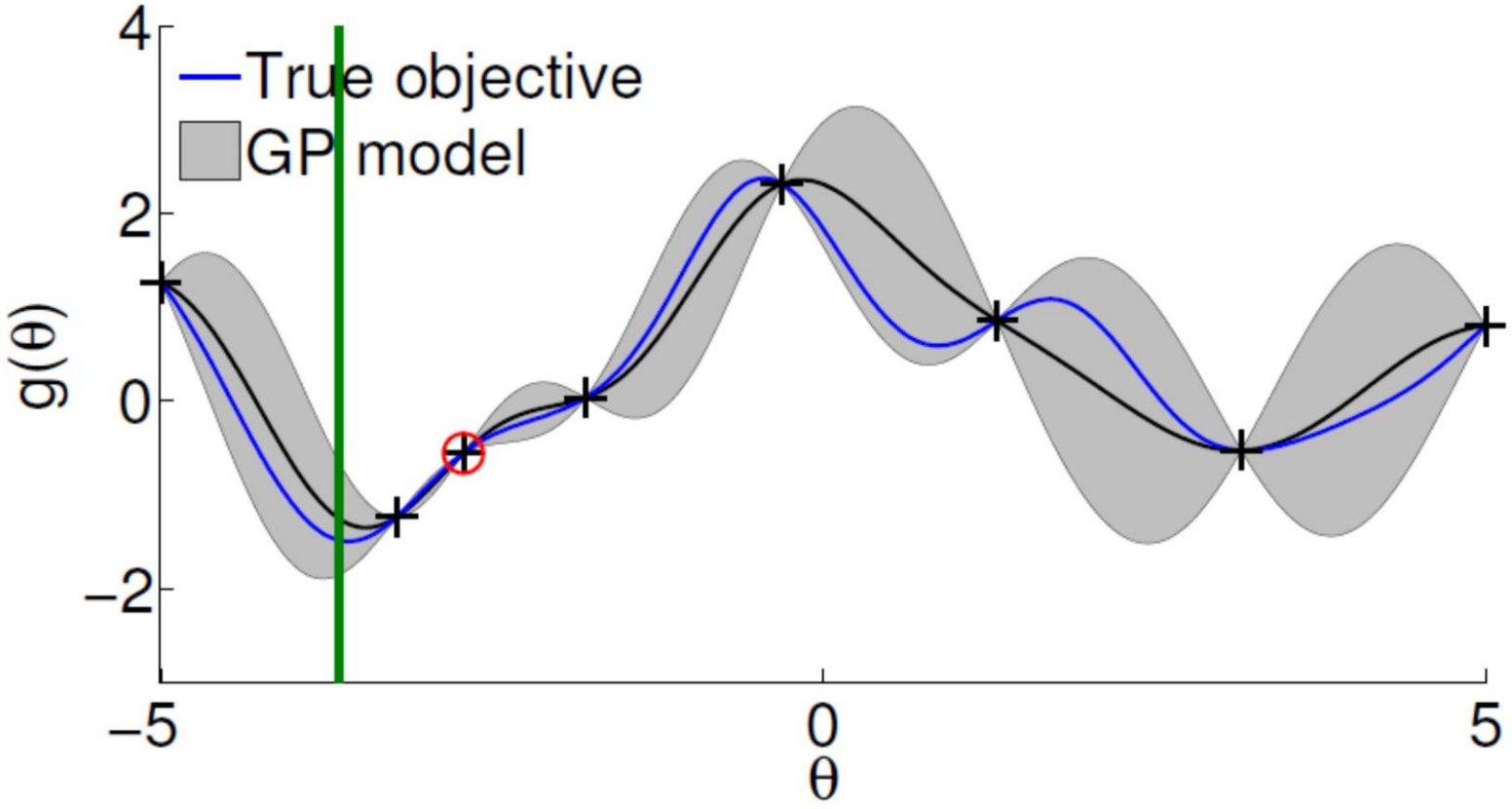
Bayesian Optimization: Illustration



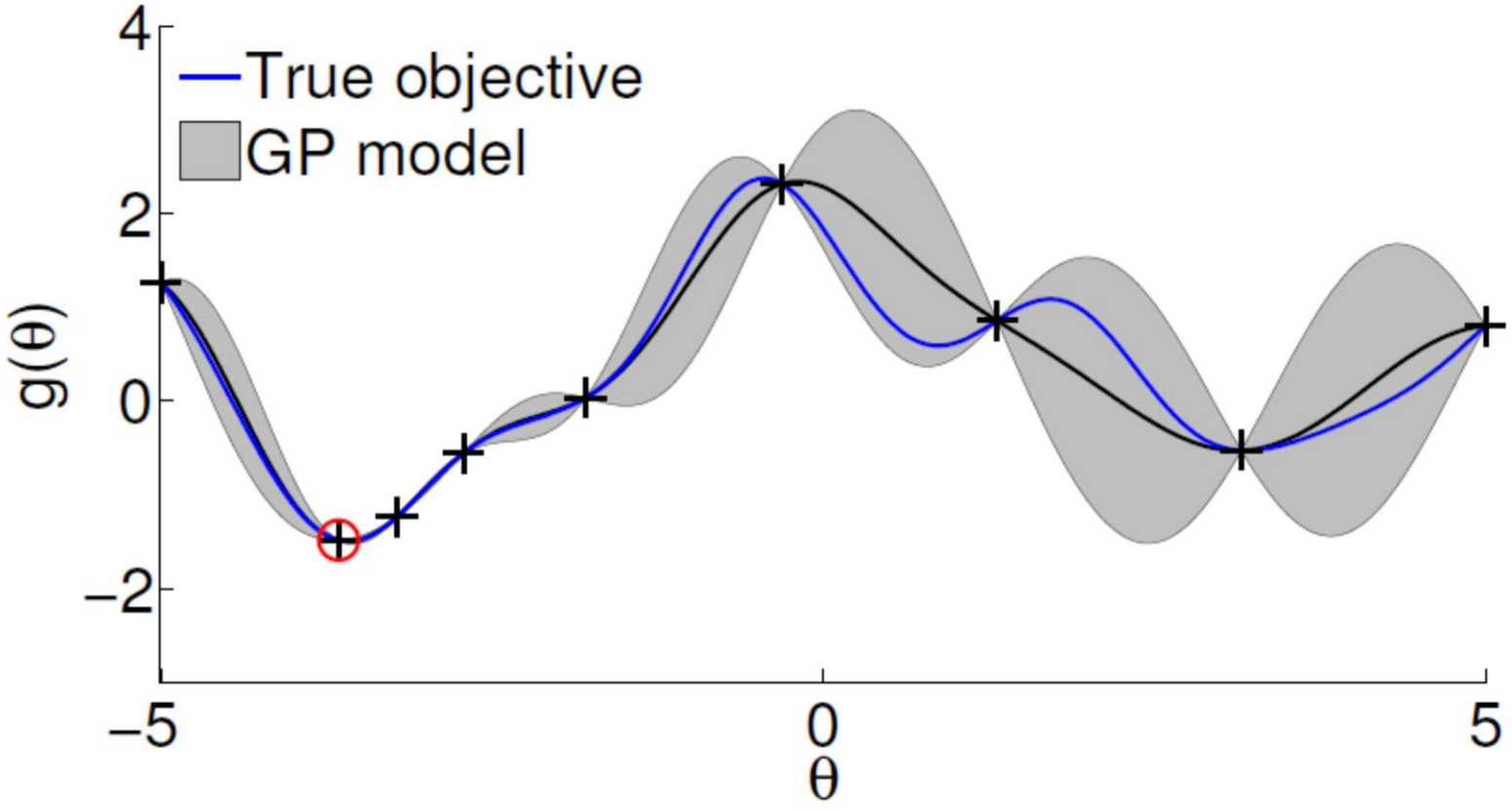
Bayesian Optimization: Illustration



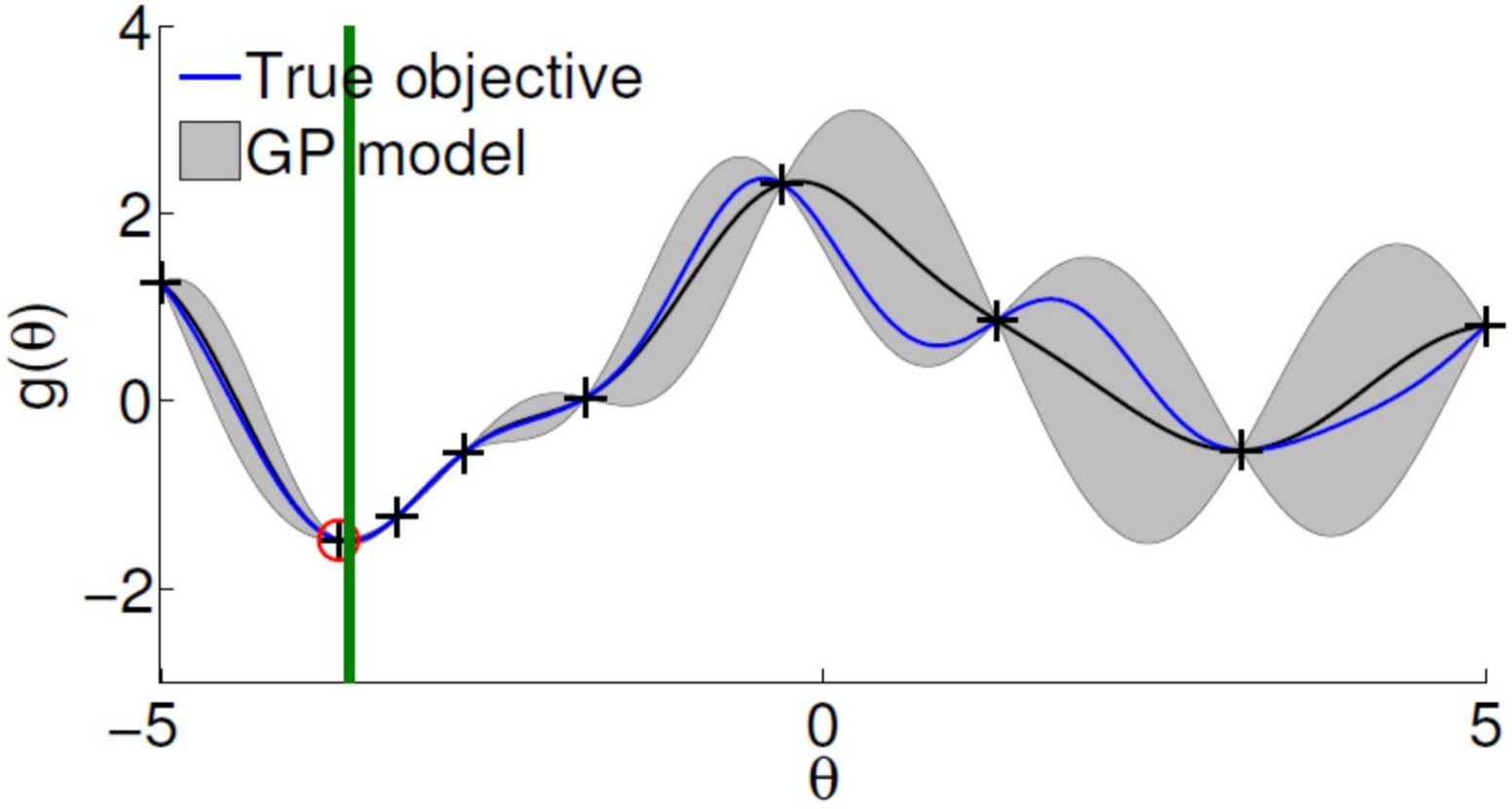
Bayesian Optimization: Illustration



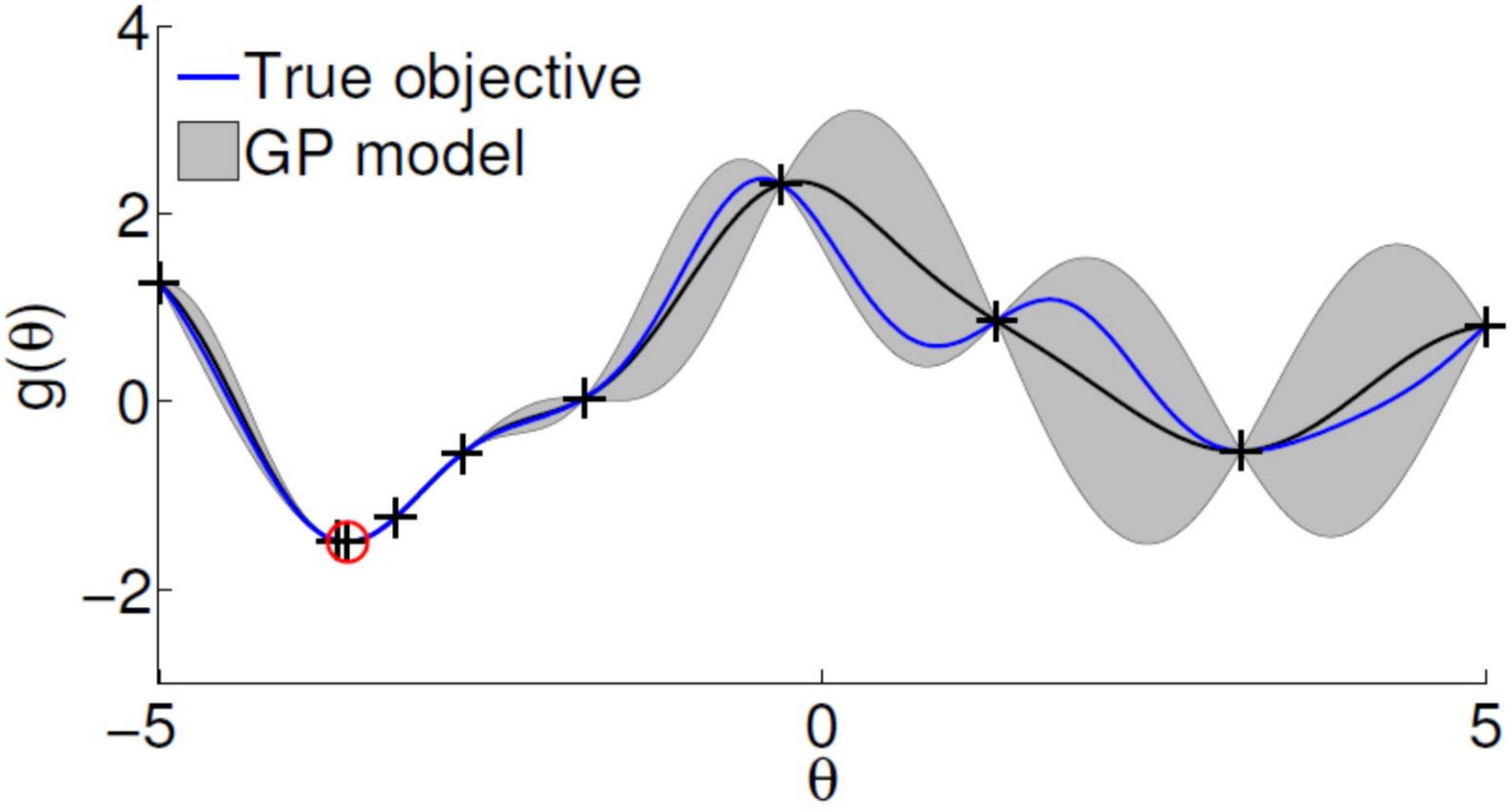
Bayesian Optimization: Illustration



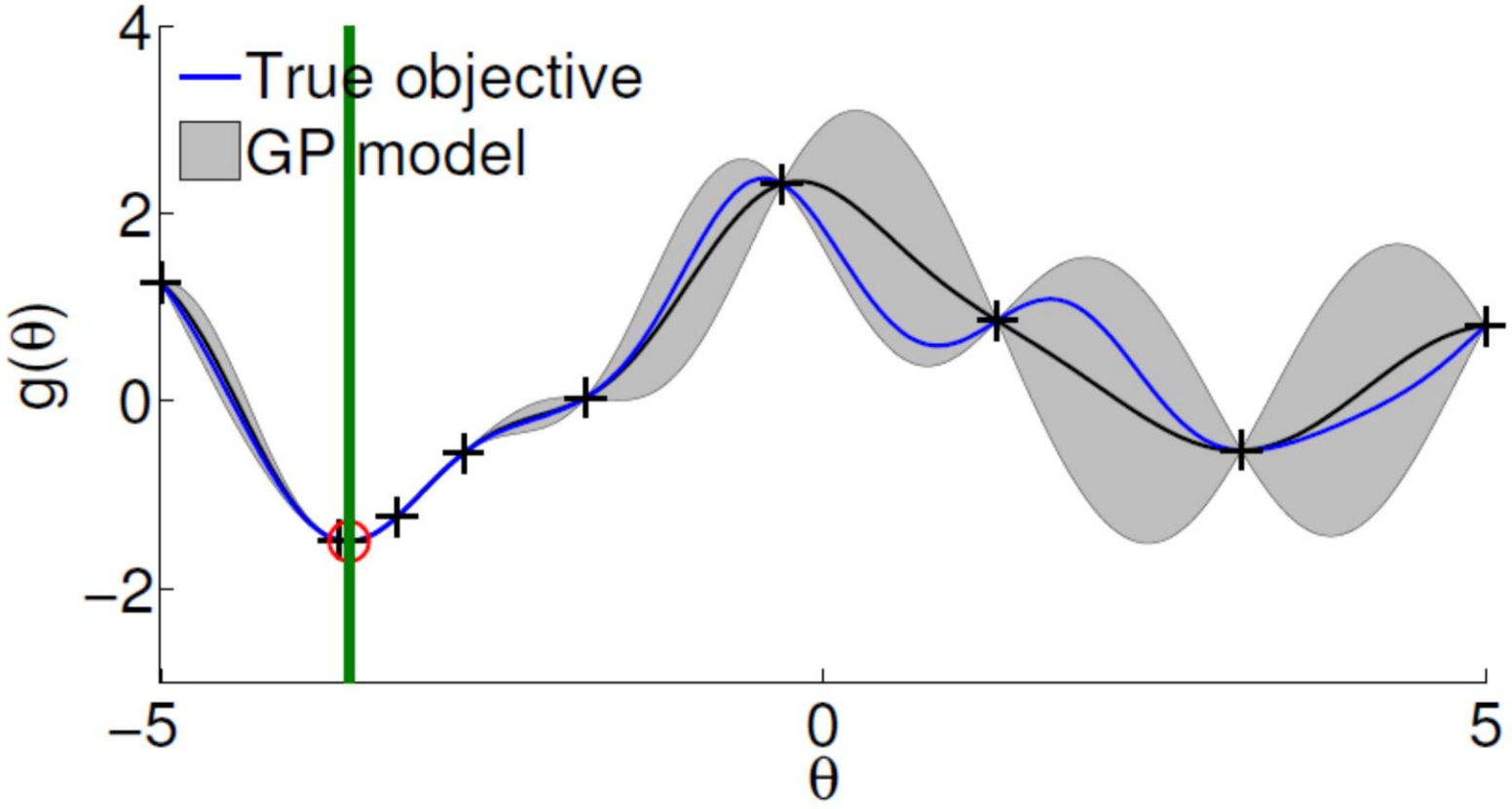
Bayesian Optimization: Illustration



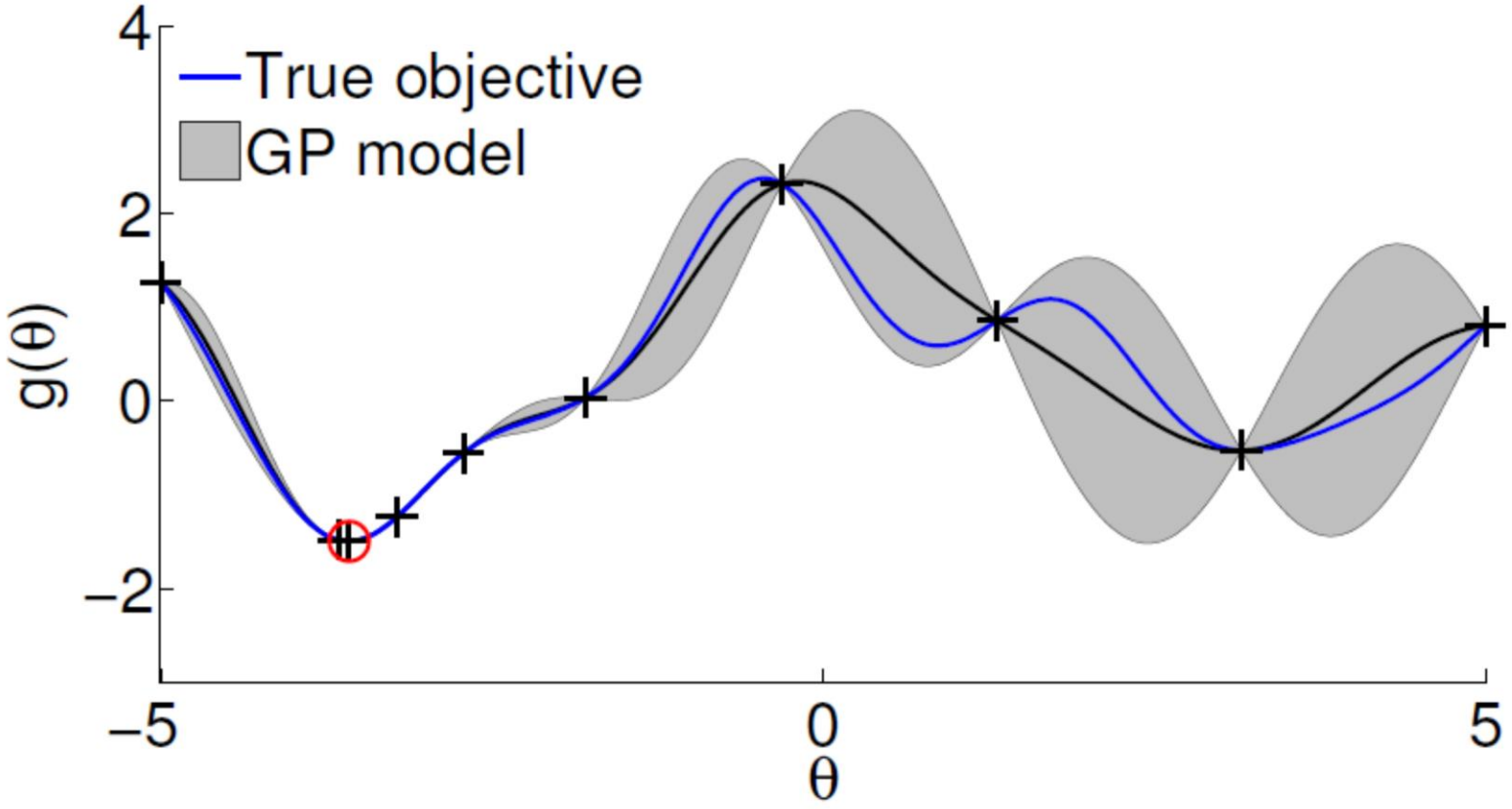
Bayesian Optimization: Illustration



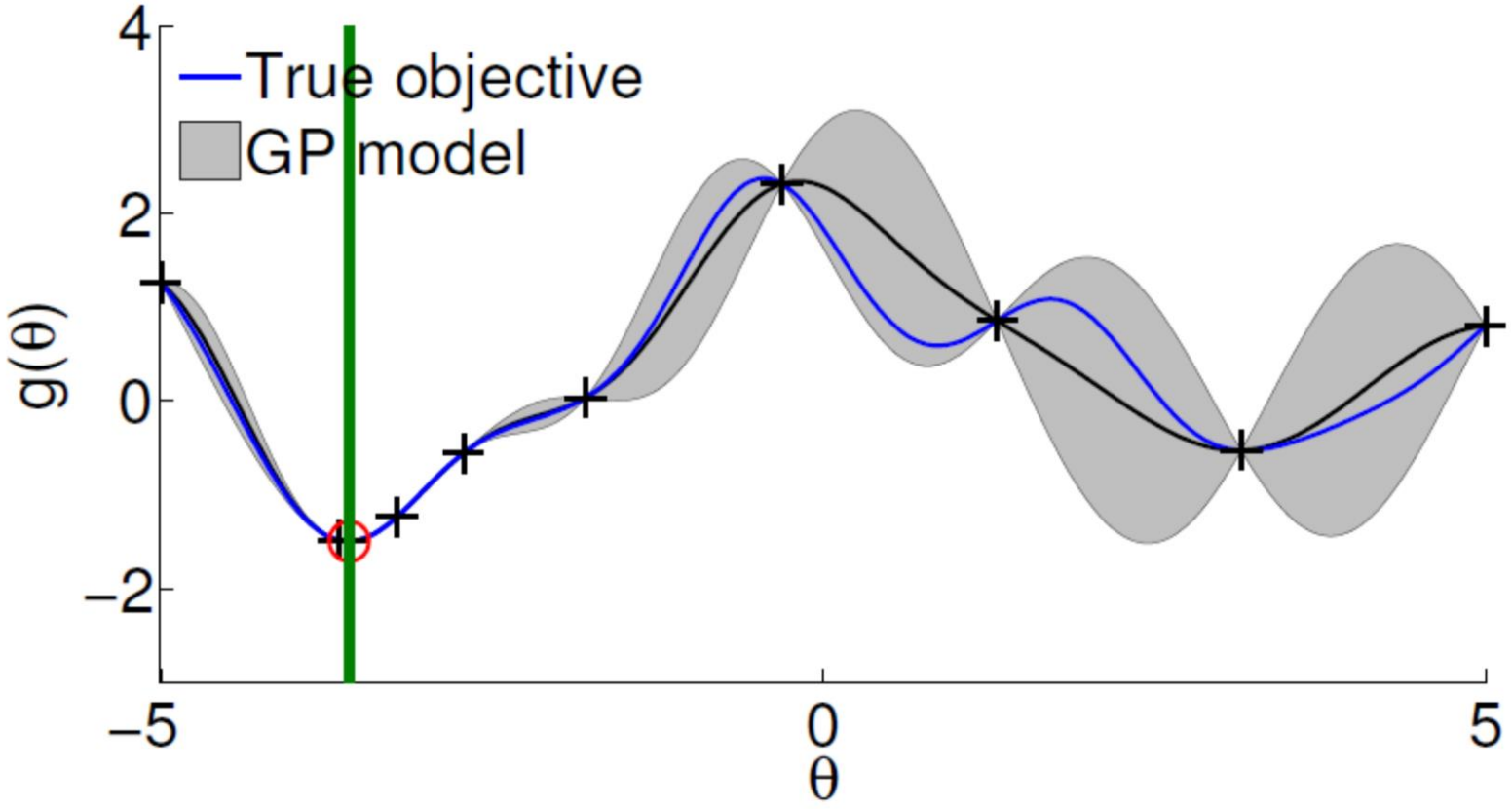
Bayesian Optimization: Illustration



Bayesian Optimization: Illustration

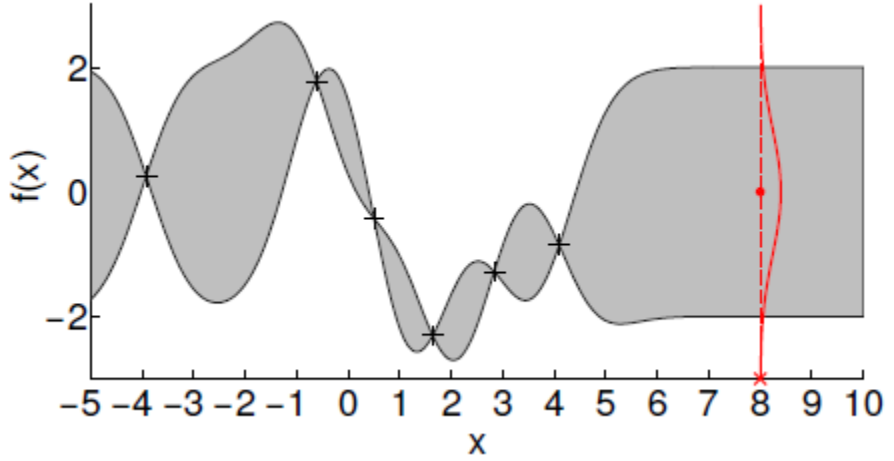


Bayesian Optimization: Illustration



Choosing the Next Point to Evaluate the True Objective: Acquisition Functions

Using Uncertainty in Global Optimization

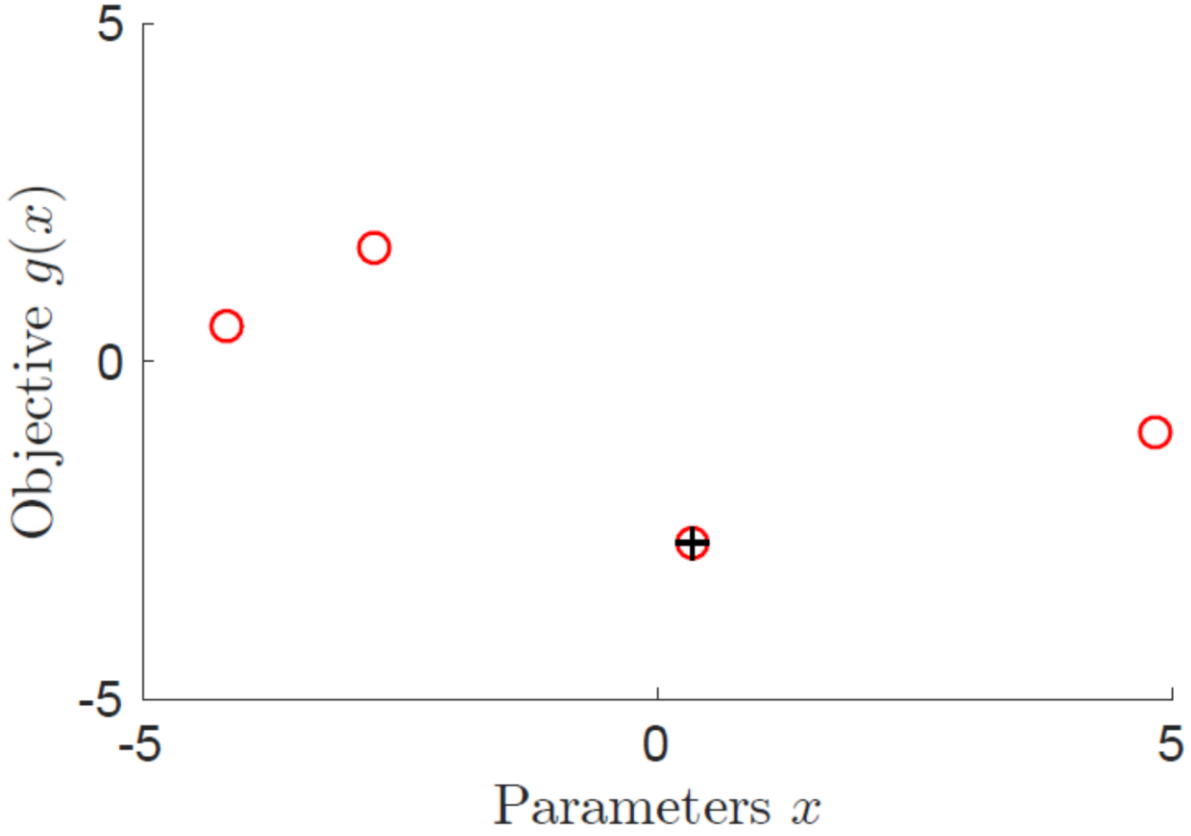


- Find a good (global) optimum
  - Need to get out of local optima
- Extrapolate from collected knowledge
- GP gives us closed-form means and variances
  - Trade off exploration and exploitation
    - **Exploration:** Seek places with high variance
    - **Exploitation:** Seek places with low mean

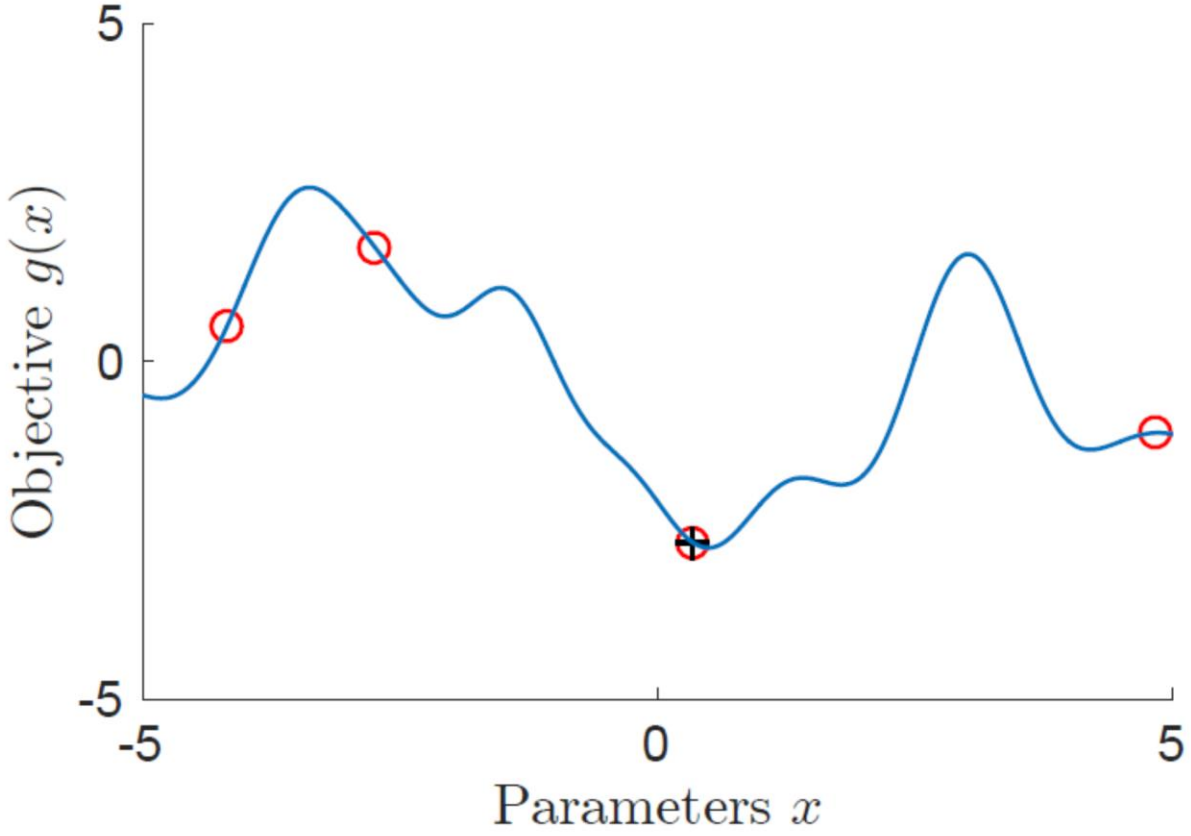
Key Steps (Pseudo-Code)

1: **Init:** Data set $\mathcal{D}_0 = \{\boldsymbol{X}_0, \boldsymbol{y}_0\}$
2: **for** iterations $t = 1, 2, \ldots$ **do**
3: Update GP using data $\mathcal{D}_{t-1}$
4: Select $\boldsymbol{x}_t = \arg\max_x \alpha(\boldsymbol{x})$ by optimizing acquisition function
5: Query true objective g at $\boldsymbol{x}_t$
6: Augment data set $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(\boldsymbol{x}_t, y_t)\}$
7: **end for**
8: **Return** best input in data set: $\boldsymbol{x}^* = \arg\min_x y(\boldsymbol{x})$

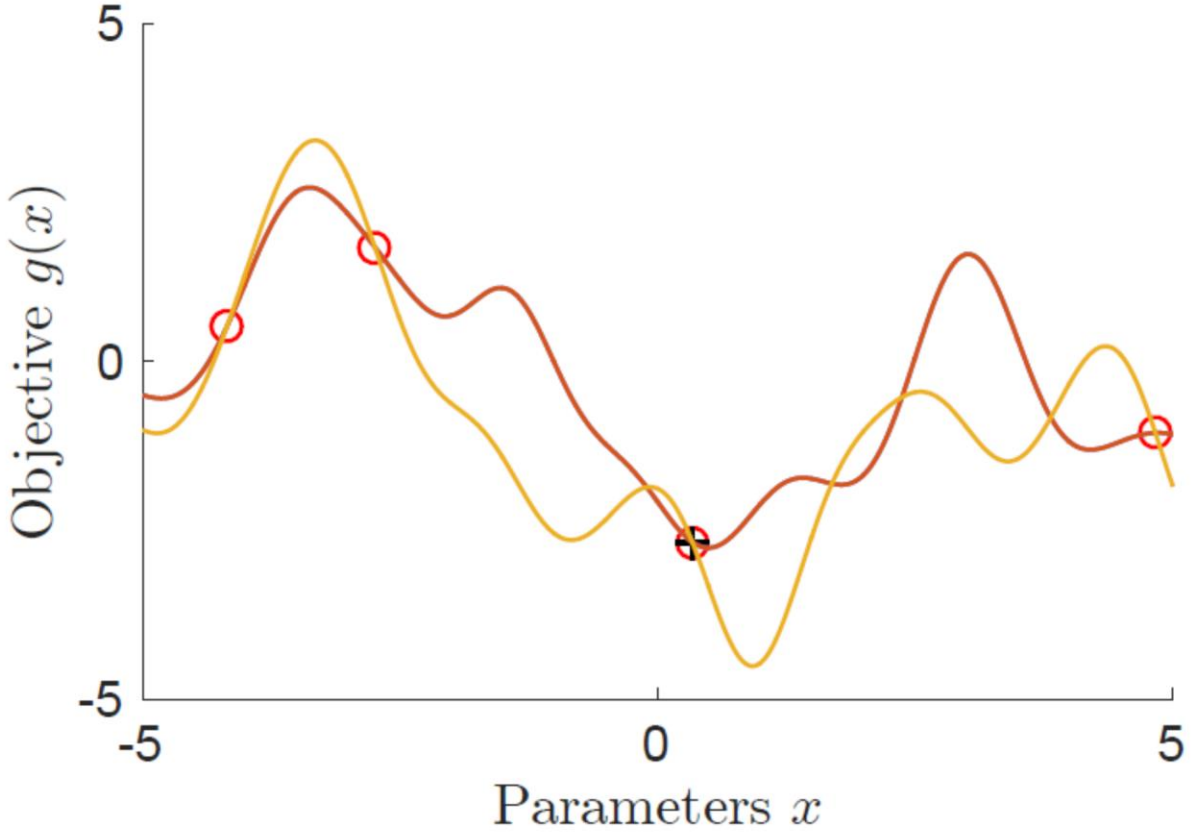
Where to Evaluate Next?



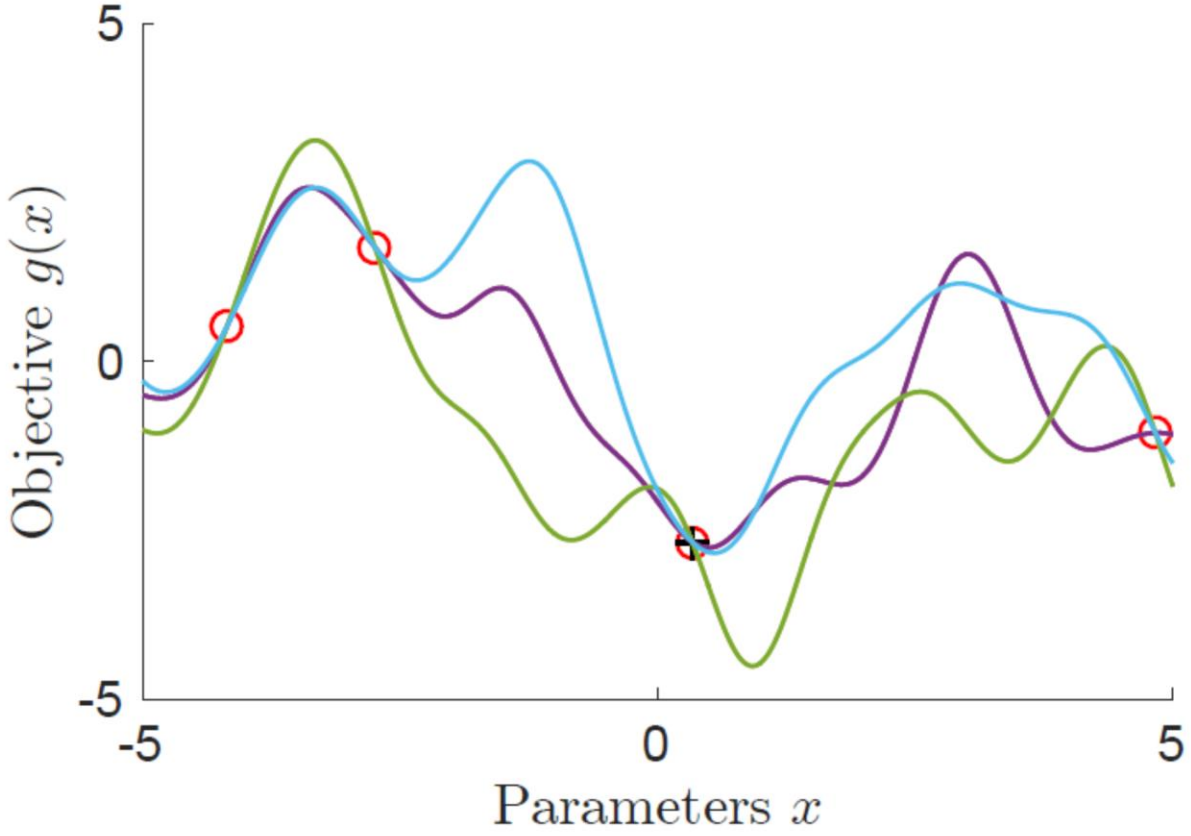
Where to Evaluate Next?



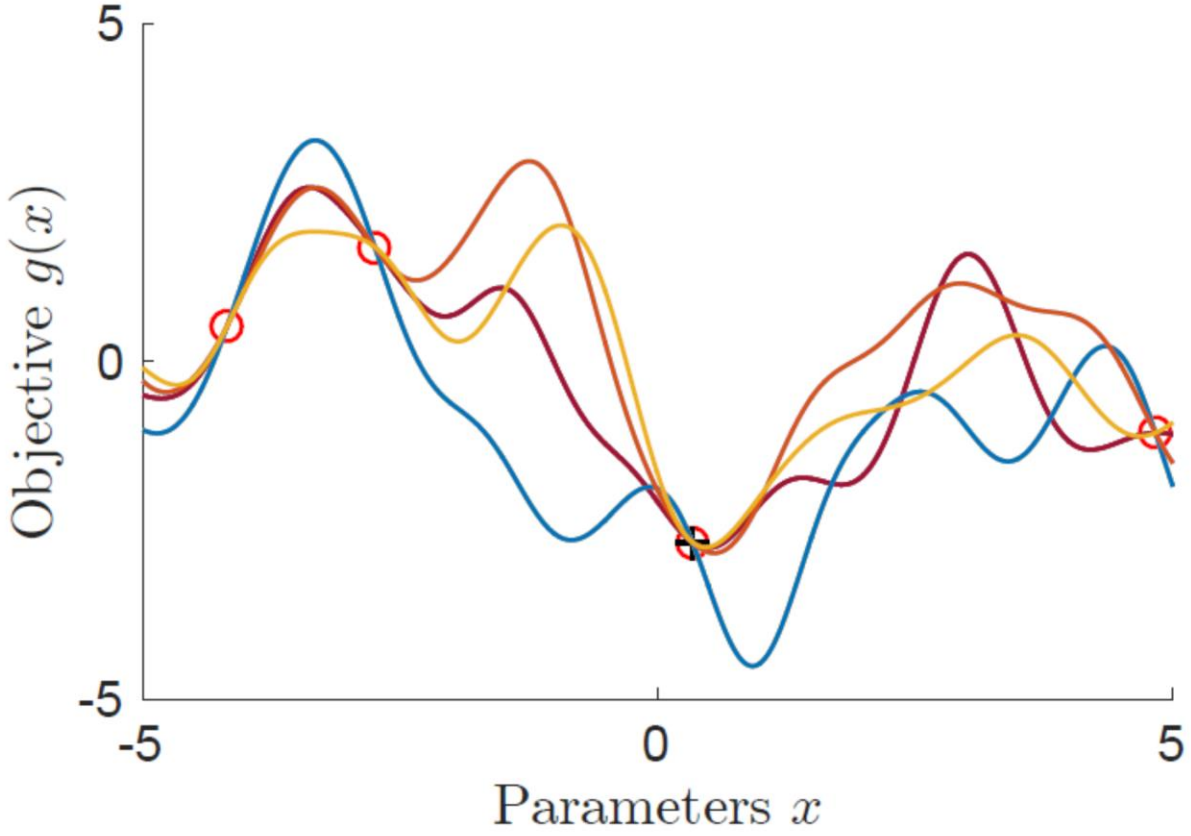
Where to Evaluate Next?



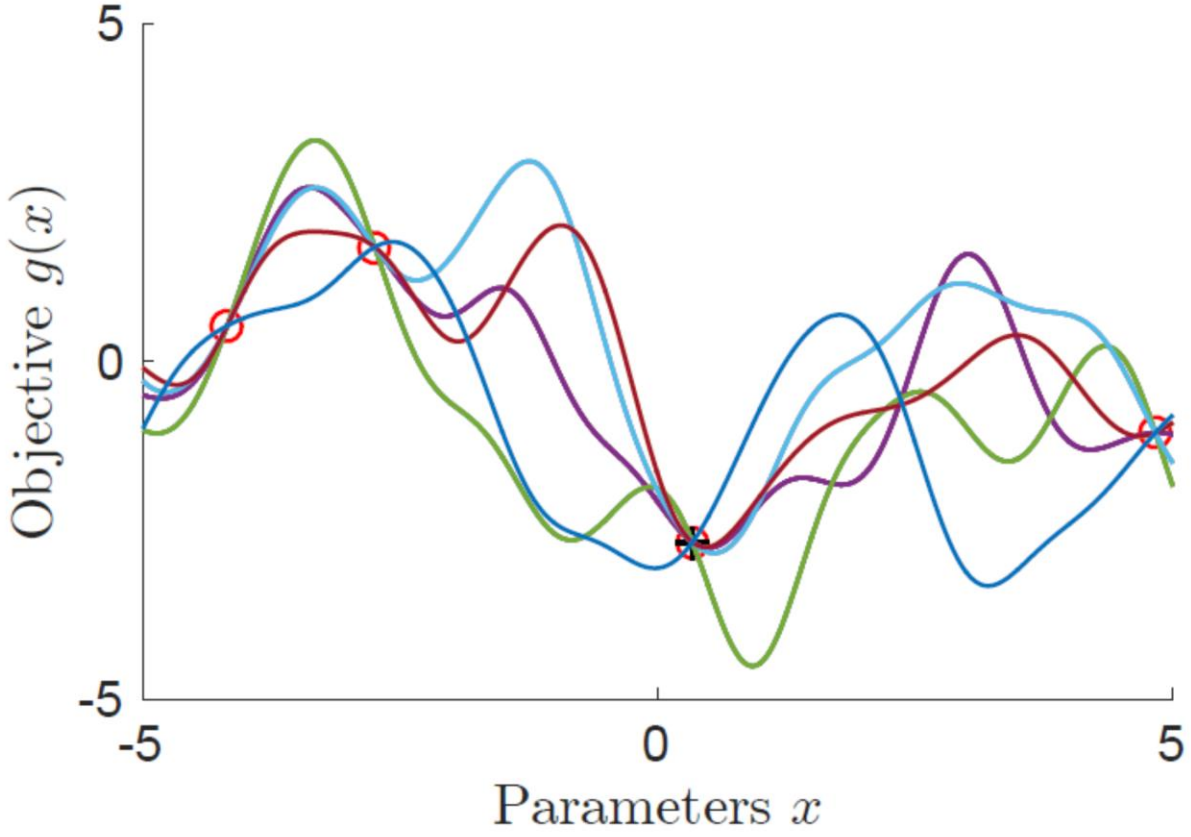
Where to Evaluate Next?



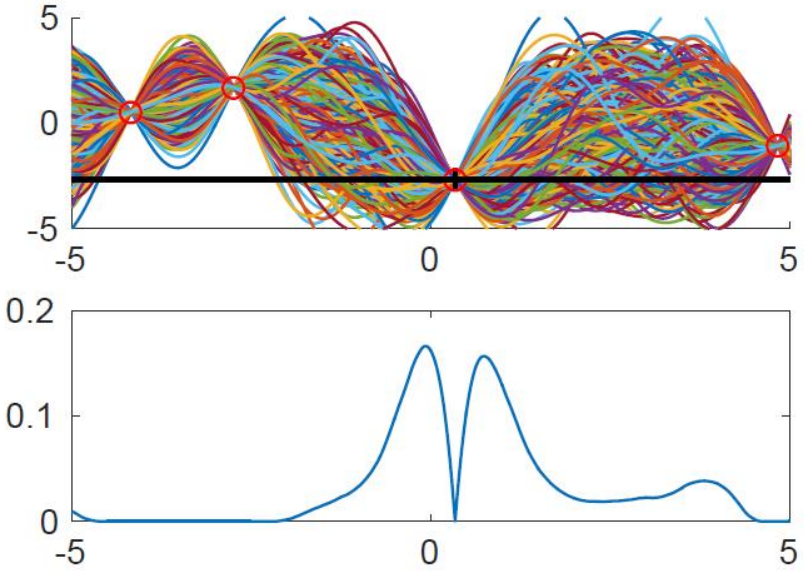
Where to Evaluate Next?



Where to Evaluate Next?



Where to Evaluate Next to Improve Most?



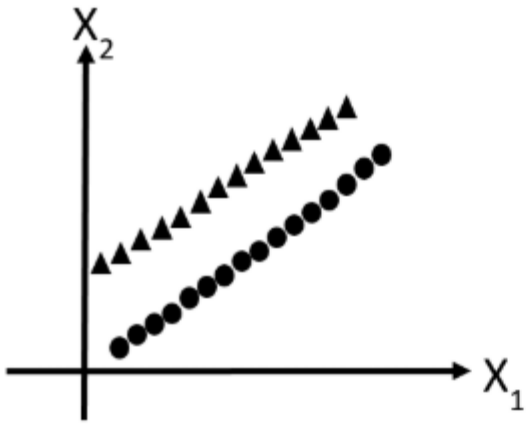
- Upper panel: Samples from a probabilistic proxy $\tilde{g}$
- Lower panel: Corresponding ***expected improvement*** over the best solution so far (black cross)
- Evaluate *g* at the maximum of the expected improvement

Summary

- Global optimization of black-box functions, which are expensive to evaluate
  - Meta-challenges in machine learning, Auto-ML
- Use a probabilistic proxy model that is cheap to evaluate and use this to suggest next experiments
- Acquisition function trade-off of exploration and exploitation

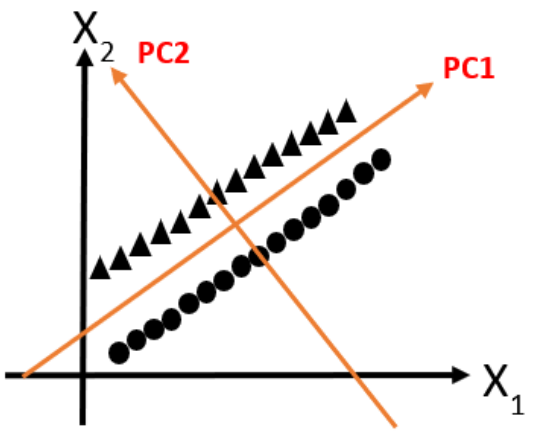
Review / Exercises

Here is a dataset with two labeled classes:



We first apply PCA on the entire dataset and obtain the first principal component (PC1) and second principal component (PC2).

If we were to use these components to classify two classes, which component is the most useful in discriminating between classes? Explain your thinking.



PC2. Although PC1 covers most of the variance in data. PC2 Provides useful information to classify two classes.

Top answer = great PCA explanation (in general)!

https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues

A fraud detection model…

- Correctly identifies 472 events (true positives);
- Incorrectly identifies 17 events (false positives);
- Correctly rejects 312159 events (true negatives);
- Incorrectly rejects 22 events (false negatives);

	TP	472	
	FP	17	
	FN	22	
	TN	312159	
a)	All P = TP+FN	494	
b)	Recall = TP / ALL P	95.5%	The model detects 95.5% of all fraudulent transactions.
c)	Precision = TP / (TP + FP)	96.5%	96.5% of predicted fraud events are actual fraudulent transactions.
d)	F1-Score = 2*Precision*Recall/(Precision+Recall)	96.0%	F1 score is the harmonic mean of porecision and recall.
e)	Specificity = TN / All N = TN / (TN + FP)	99.99%	99.99% of non-fraudulent events were correctly rejected.

1. How many actual frauds are there in the data set?
2. Calculate recall. What does this number signify?
3. Calculate precision. What does this number signify?
4. Calculate the F1-score. What does this number signify?
5. Calculate specificity. What does this number signify?
6. If you were asked to further improve the model by hyperparameter optimization and had to choose a single metric to maximize, which of the above would you choose (recall vs precision vs F1-score vs specificity)? Why?

f) If you were asked to further improve the model by hyperparameter optimization, and had to choose a single metric to maximize, which of the above would you choose (recall vs precision vs F1-score vs specificity)? Why? You can get perfect recall by having a model simply classify all events as positive (but then it's a rather useless model!). Conversely, you could have a great precision, but be missing many fraudulent cases, so that's not great either. Specificity isn't very useful, as the focus is on the wrong class (we care about frauds!). **F1-Score is the (single) best metric here**, as it's a compromise between recall and precision. A higher F1-score will mean a better model, typically.

1. What is the difference between multiclass classification and binary classification?
2. What is the shape of weights in both cases?
3. In binary linear classification, we used the linear model $y = w^T x + b$. What was the problem in using a squared error loss $L(y,t) = \frac{1}{2}(y-t)^2$?
4. Is it possible to solve that problem using an absolute value loss $L(y,t) = |y-t|^2$?

a. Two classes vs multiple classes

b. Binary (M x 1), Multiclass (M x # of classes).

c. It hates when you make predictions with high confi

d. No, it still has the same problem.

Ie high-confidence outliers are penalized heavily

Penalizing wrong answers – but doing so with a loss that is fair + appropriate for binary linear classification – motivates use of logistic function, to squash values between 0 and 1 .

Linear classification – courtesy of Roger Grosse:

https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L03%20Linear%20Classifiers.pdf

http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L04%20Training%20a%20Classifier.pdf

Loss function comparison articles:

https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/

https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

Consider this binary classification problem:

x_1	x_2	x_3	t
0	0	0	0
0	1	0	1
0	1	1	0
1	0	0	0
1	1	1	1

Your job is to find a linear classifier with weights w_1,w_2,w_3 and b which correctly classifies all of these training examples.

1. Give the set of linear inequalities the weights and bias must satisfy.
2. Give a set of the weights and bias that correctly classifies all the training examples, or provide a mathematical proof that finding such parameters is not possible.

EQ1 b<0
EQ2 b+w2>0
EQ3 b+w2+w3<0
EQ4 b+w1<0
EQ5 b+w1+w2+w3>0

(b)

b	w1	w2	w3
-0.11	0.1	0.111	-0.02

Write a Numpy code to implement a 1-nearest-neighbour classifier. Your inputs are:

- Training data $\boldsymbol{X}_{NxD}$: where N is the number of samples and D is the number of features.
- Labels $\boldsymbol{Y}_{Nx1}$: contains the corresponding integer label for each training example. (You may assume the labels are integers from 1 to K.)
- Query (test) vector $\boldsymbol{Q}_{1xD}$: you will return the predicted class for this vector.

```
[42] import numpy as np
```

```
[43] #Initialize parameters

     N = 5 #Number of samples
     D = 3 #Number of features
     K = 2 #Number of classes
```

```
[44] #Training data
     X = np.random.rand(N, D)
     X
```

```
array([[0.08254086, 0.41430511, 0.26529934],
       [0.297247  , 0.58188958, 0.55651379],
       [0.4714516 , 0.65322955, 0.7422511 ],
       [0.1851905 , 0.22224111, 0.30776268],
       [0.27937441, 0.86849952, 0.41817858]])
```

```
[45] #Class labels
     Y = np.random.random_integers(1, K, 5)
     Y
```

```
#Query (test) vector
Q = np.random.rand(1, D)
Q
```

```
array([[0.87814508, 0.1491474 , 0.32095486]])
```

```
[47] #Square of differences
     squares = (X-Q)**2
     squares
```

```
array([[6.32986078e-01, 7.03086120e-02, 3.09753774e-03],
       [3.37442589e-01, 1.87265793e-01, 5.54880084e-02],
       [1.65399592e-01, 2.54098821e-01, 1.77490519e-01],
       [4.80186060e-01, 5.34269014e-03, 1.74033762e-04],
       [3.58526316e-01, 5.17467479e-01, 9.45245132e-03]])
```

```
[48] #Distance
     dist = np.sum(squares, axis=1)
     dist
```

```
array([0.70639223, 0.58019639, 0.59698893, 0.48570278, 0.88544625])
```

```
[49] #Minimum distance element
     min = np.argmin(dist)
     min
```

```
3
```

```
[50] #Predicted class
     Y[min]
```

```
1
```

Write a Numpy code to implement a 1-nearest-neighbour classifier. Your inputs are:

- Training data $\boldsymbol{X}_{NxD}$: where N is the number of samples and D is the number of features.
- Labels $\boldsymbol{Y}_{Nx1}$: contains the corresponding integer label for each training example. (You may assume the labels are integers from 1 to K.)
- Query (test) vector $\boldsymbol{Q}_{1xD}$: you will return the predicted class for this vector.

```
dist = np.sum((X - x_query)**2, axis=1)
return y[np.argmin(dist)]
```

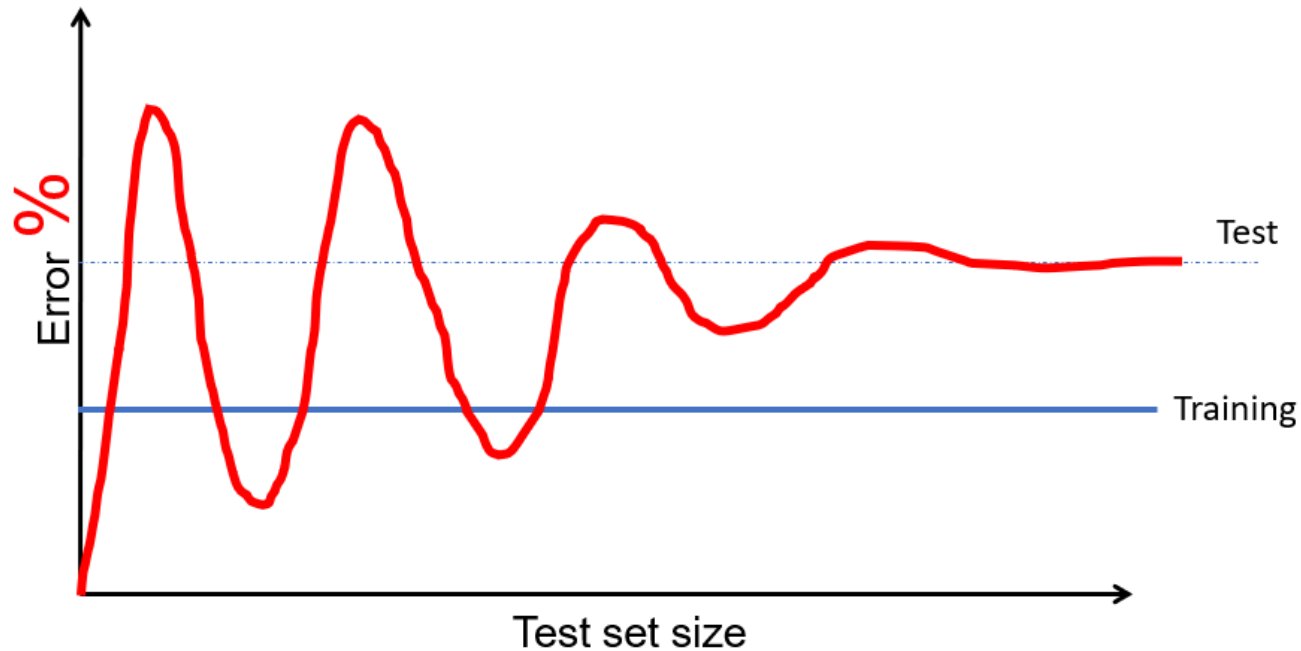
Compute covariance matrix for the dataset below:

Datapoint	X1	X2
1	1	-1
2	0	0
3	1	0
4	0	-1
5	1	-1
6	1	-1
7	0	0
8	0	0

$$S = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})'$$

	X1	X2
X1	0.285714	-0.142857
X2	-0.142857	0.285714

Plot a graph where x-axis is the ***test set size*** and y-axis show (i) the ***training error*** and (ii) the ***test error***. Assume a well-trained, typical machine learning model with fixed complexity and a fixed training set. Describe your graph.



Using row echelon technique, show if the vectors below are linearly independent or not.

A=[1,0,1,-1,0] ; B=[2,-1,0,1,0] ; C=[-2,0,1,0,0]; D=[1,-2,0,1,1]

$$\begin{bmatrix} 1 & 2 & -2 & 1 \\ 0 & -1 & 0 & -2 \\ 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ldots\ldots \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(Linearly independent)

All column vectors are linearly independent if and only if all columns are pivot columns. If there is at least one non-pivot column, the columns (and, therefore, the corresponding vectors) are linearly dependent.

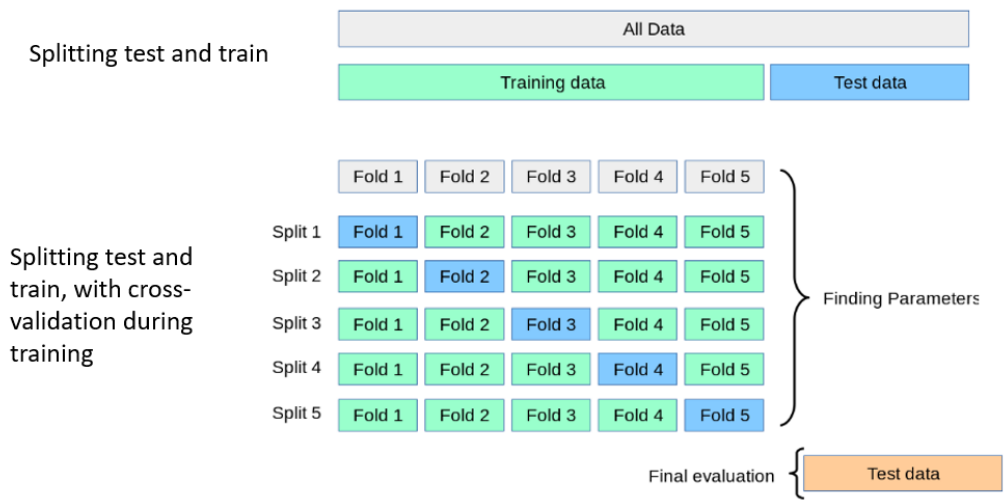
Which of the following statements is true?

a) In *k*-fold cross-validation, training data is divided into *k folds*, then the model is trained on *k* folds and validated on 1 fold, for a total of *k-1* times (splits).

b) In *k*-fold cross-validation, training data is divided into *k folds*, then the model is trained on *k* folds and validated on 1 fold, for a total of *k* times (splits).

c) In *k*-fold cross-validation, training data is divided into *k folds*, then the model is trained on *k*-1 folds and validated on 1 fold, for a total of *k* times (splits).

d) In *k*-fold cross-validation, training data is divided into *k folds*, then the model is trained on *k*-1 folds and validated on 1 fold, for a total of *k-1* times (splits).



When performing PCA, the goal is to accomplish which of the following?

a) Maximize the variance of the primary components and maximize the residuals.
b) Minimize the variance of the primary components and minimize the residuals.
c) Maximize the variance of the primary components and minimize the residuals.
d) Minimize the variance of the primary components and maximize the residuals.

Is the matrix $A = \begin{pmatrix} 1 & 2 & -1 \\ 1 & 0 & 3 \\ -2 & -4 & 2 \end{pmatrix}$ invertible?

For $n = 3$ (known as Sarrus' rule),

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{21}a_{32}a_{13} + a_{31}a_{12}a_{23}$$
$$- a_{31}a_{22}a_{13} - a_{11}a_{32}a_{23} - a_{21}a_{12}a_{33}\,.$$

=1*0*2+1*(-4)*(-1)+(-2)*2*3-(-2)*0*(-1)-1*(-4)*3-1*2*2=0+4-12+0+12-4=0

No -> determinant is 0

We have $x = [1,1,2]^T \in \mathbb{R}^3$ and $y = [1,0,3]^T \in \mathbb{R}^3$. What is the angle between vectors?

Example 3.6 (Angle between Vectors)
Let us compute the angle between $x = [1,1]^\top \in \mathbb{R}^2$ and $y = [1,2]^\top \in \mathbb{R}^2$; see Figure 3.5, where we use the dot product as the inner product. Then we get

$$\cos\omega = \frac{\langle x, y\rangle}{\sqrt{\langle x, x\rangle \langle y, y\rangle}} = \frac{x^\top y}{\sqrt{x^\top x y^\top y}} = \frac{3}{\sqrt{10}}, \tag{3.26}$$

and the angle between the two vectors is $\arccos(\frac{3}{\sqrt{10}}) \approx 0.32\,\text{rad}$, which corresponds to about 18°.

$$\cos(w) = \frac{x^T y}{\sqrt{x^T x y^T y}} = \frac{7}{\sqrt{6*10}} = 25.4\ degrees$$

Calculate the Jacobian $\boldsymbol{J_F}(x_1, x_2, x_3)$ of the function $\boldsymbol{F}: \mathbb{R}^3 \to \mathbb{R}^4$, which has components:

$$y_1 = x_1 + 2x_3^2 \; ; \; y_2 = x_1 \sin x_2 \; ; \; y_3 = x_2 \exp(x_3) \; ; \; y_4 = x_1 + x_2$$

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \frac{\partial y_1}{\partial x_3} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \frac{\partial y_2}{\partial x_3} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} & \frac{\partial y_3}{\partial x_3} \\ \frac{\partial y_4}{\partial x_1} & \frac{\partial y_4}{\partial x_2} & \frac{\partial y_4}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 4x_3 \\ \sin(x_2) & x_1\cos(x_2) & 0 \\ 0 & \exp(x_3) & x_2\exp(x_3) \\ 1 & 1 & 0 \end{bmatrix}$$

Consider functions $f: \mathbb{R}^3 \to \mathbb{R}^1$ and $\boldsymbol{x}: \mathbb{R}^1 \to \mathbb{R}^3$, where:

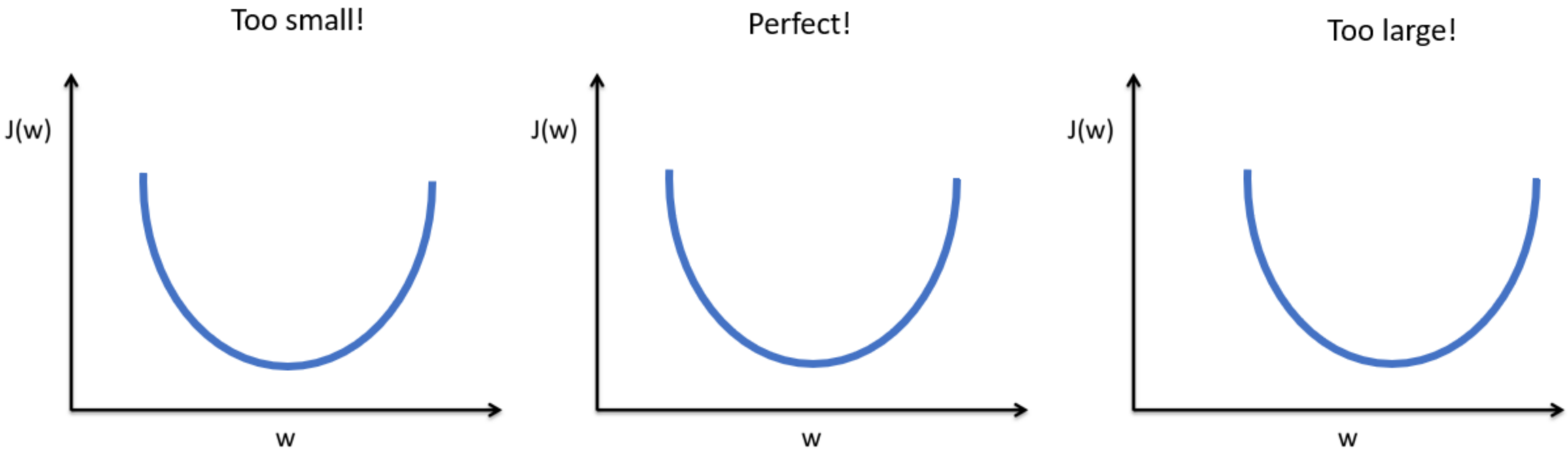
$$f(\boldsymbol{x}) = x_1 + 2x_2^2 + x_3 \; ; \boldsymbol{x}(t) = \begin{bmatrix} t \\ 3t \\ \exp(t) \end{bmatrix}$$

Calculate the gradient $\frac{df}{dt}$ using the chain rule.

$$\frac{df}{dt} = \frac{df}{dx}\frac{dx}{dt} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \\ \frac{\partial x_3}{\partial t} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 4(3t) & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ exp(t) \end{bmatrix} = 1 + 36t + exp(t)$$

What is a learning rate? Explain what happens if it is set too high or too low (you may use a schematic/drawing if helpful).



In general, is using a gradient descent algorithm a good choice for finding optimal hyperparameters? Why or why not?

Gradient descent can work for some specific machine learning problems, but not all. Gradient descent requires a smooth, convex function to optimize… Bayesian optimization often used to optimize hyperparameters – no need to calculate gradient.