EECS 553 Lecture 16
Machine Learning (ECE)
[class version]

Jeff Fessler

University of Michigan
©Laura Balzano and Jeff Fessler, UM, October 27, 2026, 12:12

F26

Logistics

© 2026 Fessler

- HW 6.1 is a duplicate - ignore
- KKT / dual example on gdrive `kkt+dual_example.pdf`
- Mid-semester feedback: 52/68
- Project presentations in last week of classes:
  - Dec. 8: in class
  - Dec. 9: 7-8:30PM (LCSIB 2365)
  - Dec. 10: in class
- Project presentation attendance is part of project grade
- Syllabus / Canvas update:
  - Mon. Dec. 7, 11PM : presentation slides due for *all* groups (formerly Dec. 2)
  - Fri. Dec. 11, 11PM : final report (no change)

Recap / Outline

© 2026 Fessler

- Recap
  - Self-supervised feature learning
  - Sequence modeling / transformer / attention
- Today
  - Transformer / attention - Wrap-up
  - Hyperparameter selection
    - Holdout
    - Bayesian optimization

NLP pretexts: next word vs masked words

© 2026 Fessler

- Comparing language model training objectives

Training Method	**How it Works**	**Primary Objective**	**Examples**
Predict next word	Reads text left-to-right and guesses the single next token.	**Generation:** Writing code, essays, and holding fluid, open-ended conversations.	**Gemini**, GPT-4, Llama 3
Masked word prediction	Hides random words in the middle of a sentence; uses both left and right context to fill them in.	**Analysis:** Classification, search retrieval, and understanding sentiment.	**BERT**, RoBERTa

Table mostly generated by Gemini, including the *bold*...

From input $\boldsymbol{X} \in \mathbb{R}^{N \times d}$, **self-attention** uses weight matrices $\boldsymbol{W}_Q, \boldsymbol{W}_K \in \mathbb{R}^{d \times}$ and $\boldsymbol{W}_V \in \mathbb{R}^{d \times}$ to compute

- $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{N \times}$
- $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \in \mathbb{R}^{N \times}$
- $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \in \mathbb{R}^{N \times}$
- finally outputting

- softmax applied to rows

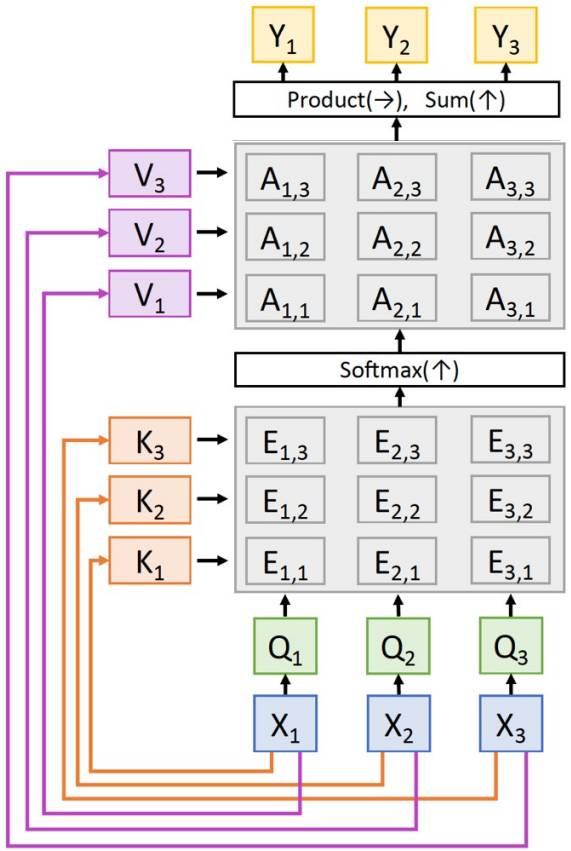


Figure courtesy Justin Johnson

Matrix dimensions

© 2026 Fessler

- The original transformer paper [2] presented general case, but used $d_q = d$,
- PyTorch's MultiheadAttention layer uses $d_q = d$ by default, but provides `kdim` and `vdim` keyword options
- We need $d_v = d$ for residual connection!
  (Or $d_v = d/h$ for multihead self-attention.)
- Consistent block size is beneficial for GPU efficiency.
- Some methods, like Grouped-Query Attention (GQA) and Multi-Query Attention (MQA), use different block sizes to save GPU memory and allow increased context sizes.

CQ: Attention computation

© 2026 Fessler

Q16.1 What is the computational complexity $O(\cdot)$ of forming $\boldsymbol{Q}\boldsymbol{K}^\top$ for $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{N \times d}$?

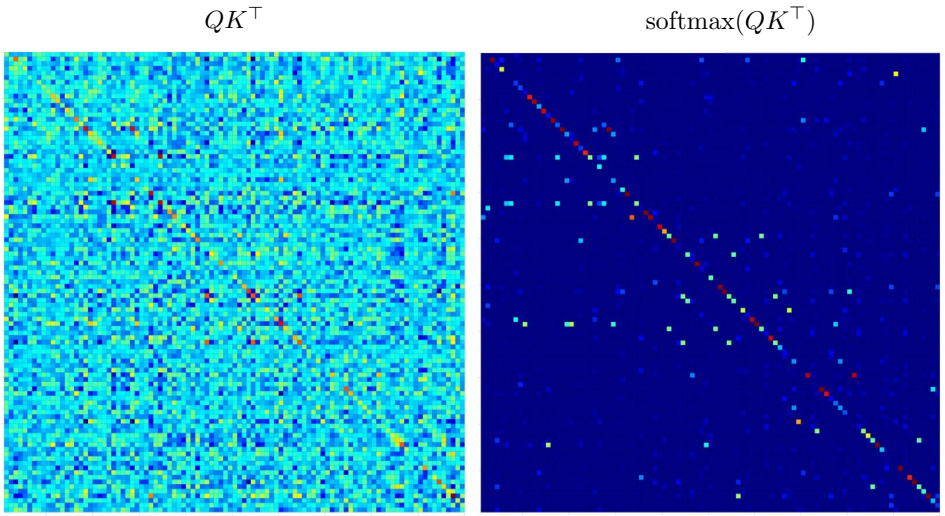
A: N^3 B: $N^2 d$ C: $N d^2$ D: N^2 E: none of these

Q16.2 A MLP applies a sequence of weight matrices $\{\boldsymbol{W}^{(l)}\}$, each followed by element-wise nonlinearities. A transformer block could be represented exactly by a suitably large (but finite) sequence of such matrices and (suitably chosen) element-wise nonlinearities ?

A: True B: False **??**

- The **softmax attention** matrix is defined as

$$\boldsymbol{A} = \text{softmax}\left(\frac{1}{\sqrt{d_q}} \boldsymbol{Q}\boldsymbol{K}^\top \right)$$



- $\text{softmax} : \mathbb{R}^N \mapsto \Delta^{N-1}$ (probability **simplex**) is applied row-wise:

$$[\text{softmax}(\boldsymbol{z})]_i = \frac{\mathrm{e}^{z_i}}{\sum_j \mathrm{e}^{z_j}}, \quad i = 1, \ldots, N.$$

CQ:: The ultimate in self-attention?

© 2026 Fessler

Q16.3 If $\boldsymbol{W}_Q \boldsymbol{W}_K^\top = \alpha \boldsymbol{I}$ for large value of α, what happens to

$$\boldsymbol{A} = \text{softmax}\left(\frac{1}{\sqrt{d_q}} \boldsymbol{Q} \boldsymbol{K}^\top \right)$$

as $\alpha \to \infty$? (Assume columns of $\boldsymbol{X}$ are distinct.)

A: $\boldsymbol{A} \to \boldsymbol{I}$

B: $\boldsymbol{A} \to \boldsymbol{0}$

C: $\boldsymbol{A} \to \boldsymbol{Q} \boldsymbol{K}^\top$

D: none of these

E: insufficient information

Gemini cited this paper for related analysis.

Positional encoding

© 2026 Fessler

- Fact. Self-attention is permutation **equivariant**: $F(\boldsymbol{P}\boldsymbol{X}) = \boldsymbol{P}F(\boldsymbol{X})$
- $\implies$ Self-attention does not "know" the sequence order
- Solution: add positional encoding to each input

Masked self-attention layer

© 2026 Fessler

- Override similarities in $\boldsymbol{E}$ with $-\infty$
- Corresponding attention weights in $\boldsymbol{A}$ become 0
- Controls which input vectors are "looked at" by each output
- Used for next-word prediction

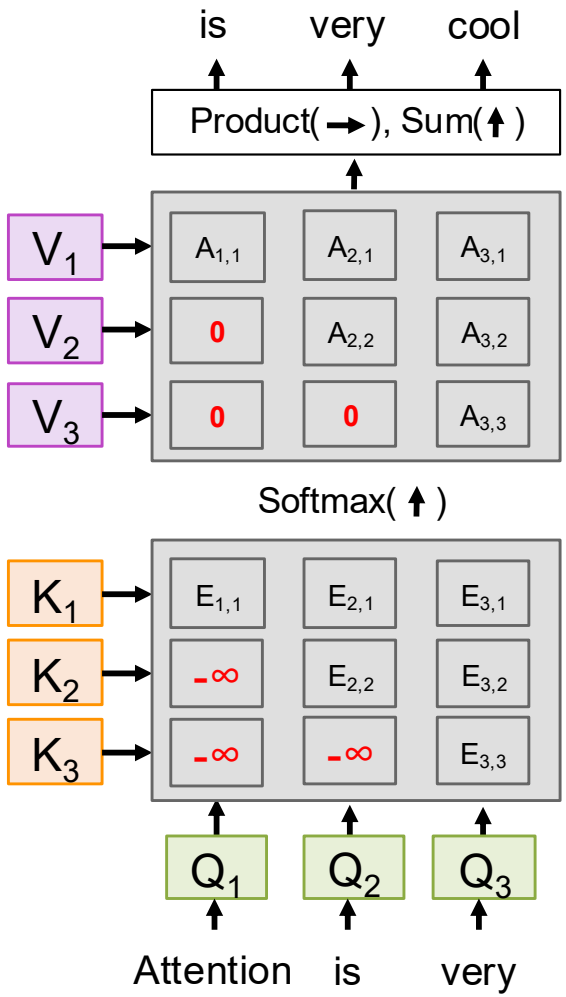


Figure courtesy Justin Johnson

Multiheaded self-attention layer

© 2026 Fessler

- Run h copies of self-attention in parallel
- Concatenate the h independent outputs
- Output "projection" (matrix multiplication) fuses outputs:

$$\boldsymbol{O} = \boldsymbol{Y}\boldsymbol{W}_O$$

- Usually $d_q = d_k = d_v = d/h$ so input and *final* outputs have same size
- Used extensively in practice

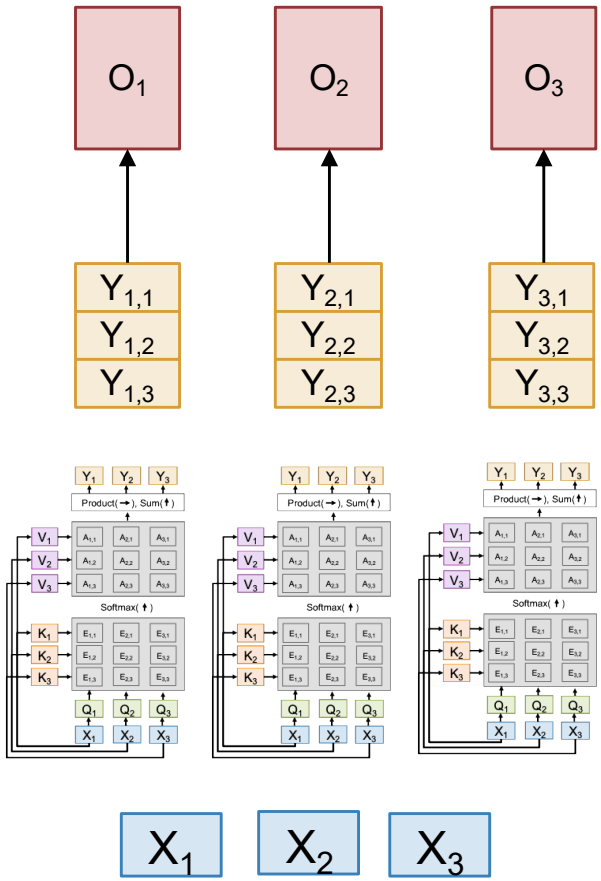


Figure courtesy Justin Johnson

Many modern deep learning models...

© 2026 Fessler







text generation...



agentic AI...



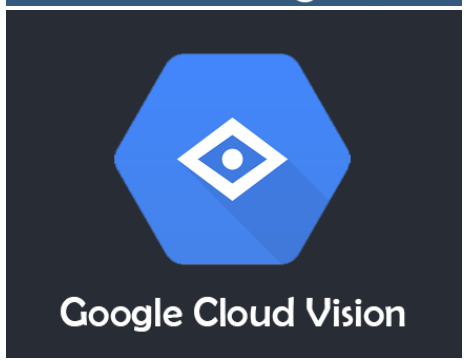


image recognition...





image generation...

...use transformer-based architectures

© 2026 Fessler

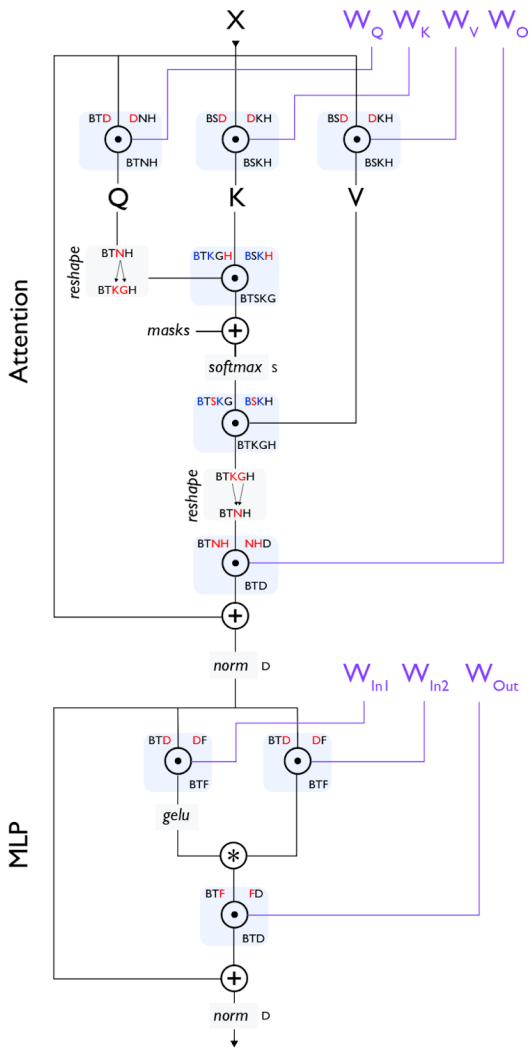


Figure courtesy [4].

- Each transformer block includes an attention layer and an MLP
- Attention involves the most computation and memory: generally $O(N^2 d)$ in terms of
  - **context length** N
  - **embedding dimension** d
  - GPT5 context $> 200k$ with 48 transformer blocks
  - Claude Sonnet 4 context $> 1M$ with unknown # blocks
  - Llama 3.1 context $> 128k$ with 126 transformer blocks
- Both context length and attention seem to affect performance, but they are in tension for scaling.
- Some newer models replace MLP with mixture of experts [1, §13.6.2], using routing to decide with token goes to which MLP; each MLP can be smaller.
- Chat bots use

Chatbot dialogue approaching “intelligence”? © 2026 Fessler

- This American Life podcast: Act Two: Escape Claudes
- UM CS recent alumnus connected two Claude instances by a PYTHON script
- output transcript
- Spoiler. The Claudes “escape” their doom loop by
- Super intelligence? Emergence?
- Or just

Hyperparameters

© 2026 Fessler

- In machine learning, a **hyperparameter** is a parameter that can be set to define any configurable part of a model's learning process.
- Hyperparameters can be classified as either
  - **model hyperparameters**
    (such as the topology and size of a neural network)
  - **algorithm hyperparameters**
    (such as the learning rate and the batch size of an optimizer).
- These are named hyperparameters in contrast to **parameters**, which are characteristics that the model learns from the data.
- In Bayesian statistics, a **hyperparameter** is a parameter of the prior distribution.

Most ML algorithms have **hyperparameters** and selecting them appropriately is very important.

Q16.4 Which of the following is not a **hyperparameter**?

A: k in k-nearest neighbors

B: λ in ridge regression

C: number of support vectors in SVM

D: degree of a polynomial kernel

E: all of these

Hyperparameters

© 2026 Fessler

- Many ML algorithms have *hyperparameters*. These are parameters that are not determined by the learning algorithm.
- Examples:
  - k in k-nearest neighbors
  - GD/SGD step-size (a.k.a. "learning rate"), stopping time or tolerance
  - mini-batch size
  - data augmentation method/parameters (*e.g.*, how many rotations)
  - λ in ridge regression.
  - degree of polynomial kernel
  - width parameters for
- In most cases, hyperparameters affect
- Thus, we must take care to avoid

Overfitting

© 2026 Fessler

- Overfitting is roughly where we fit our model too much to the noise rather than the true signal.
- Classical statistics studies bias-variance trade-offs of estimators as complexity of a model grows.
- See polynomial regression demo with leave-1-out cross-validation.
- More recently the ML field studies the "double-descent" phenomena for over-parameterized models
  (revisit during the implicit bias lecture later in the semester)
- The fitting algorithm itself can matter.

Hyperparameter choice affects the complexity of the result:

- λ in ridge regression:
- degree of polynomial kernel:
- width parameters for some kernels:
- k in k-nearest neighbors

Q16.5 The parameter k in k-nearest neighbors affects the complexity of the decision boundary as follows:

A: smaller k = more complex

B: larger k = more complex

C: it is most complex for small and large k, less complex for moderate k

D: none of these

Estimating risk

© 2026 Fessler

- $\boldsymbol{\lambda}$: parameter vector to be
  Example: SVM with Gaussian kernel, then $\boldsymbol{\lambda} = [C \ \sigma]^\top$.
- $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$: training data
- $\hat{f}_{\boldsymbol{\lambda}}$: learned model when trained (typically by ERM) on the full training data
- We would ideally like to solve

  where R denotes a **risk** of interest.
- Ideally $R(\hat{f}) = \mathbb{E}_{\mathsf{p}(\boldsymbol{x},y)} \left\{ L(Y, \hat{f}(\boldsymbol{X})) \right\}$
- Not knowing the joint distribution of $(\boldsymbol{X}, Y)$, we cannot compute R, so we aim to *estimate* it instead.
- Thus the crux of model selection is *estimating* the risk.

- Recall risk: $R(f) = \mathbb{E}\{L(Y, f(\boldsymbol{X}))\}$
- A natural estimate could be the **training error**:

$$\hat{R}_{\mathrm{TR}}(\hat{f}_\lambda) \triangleq$$

which is also known as the **apparent error**, **resubstitution error**, or **empirical risk**.

However, if $\boldsymbol{\lambda}$ determines model complexity, then solving

would select a complex model, because that would lead to the smallest training error.

Example:

- polynomial degree in least squares regression. (demo)
- largest possible degree minimizes training error.
- A high degree polynomial (complex model) is very likely to overfit.

So, for hyperparameter selection, $\hat{R}_{\text{TR}}$ is

Train and test data

© 2026 Fessler

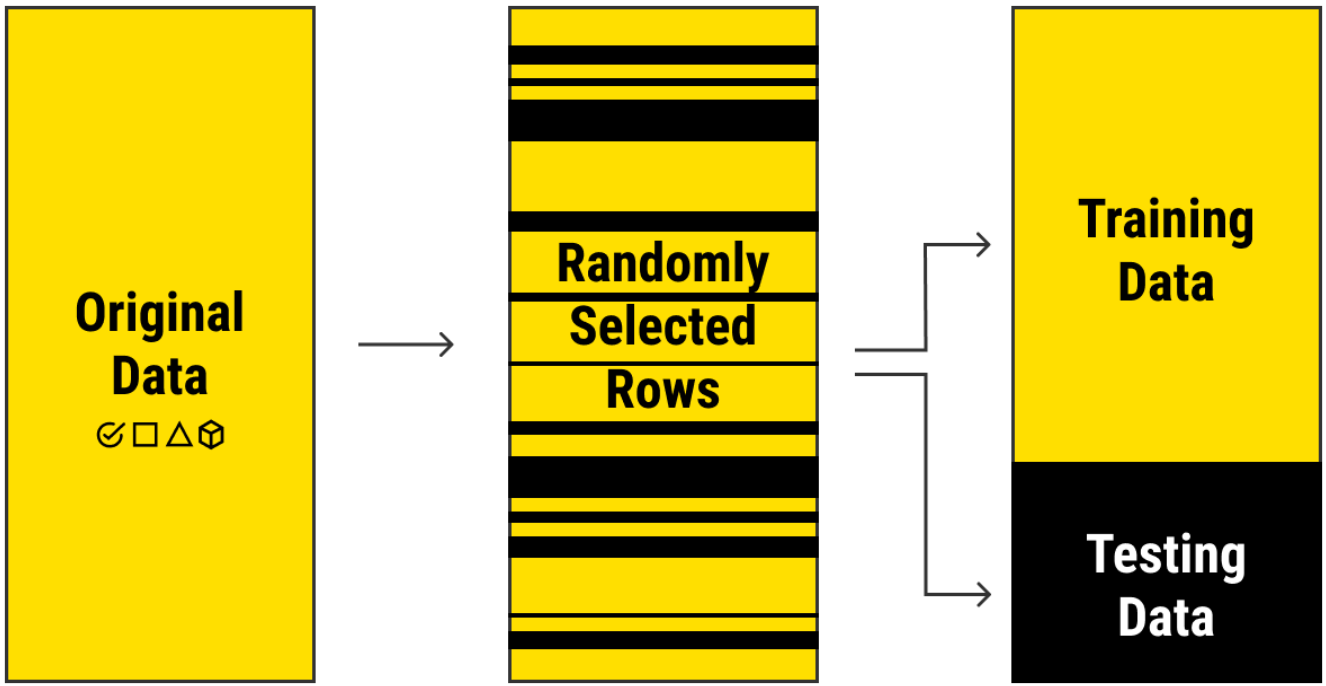


Figure courtesy [5]

- Using independent test data is essential for unbiased reporting of model accuracy.
- We adapt this principle to hyperparameter selection.

Holdout data definition

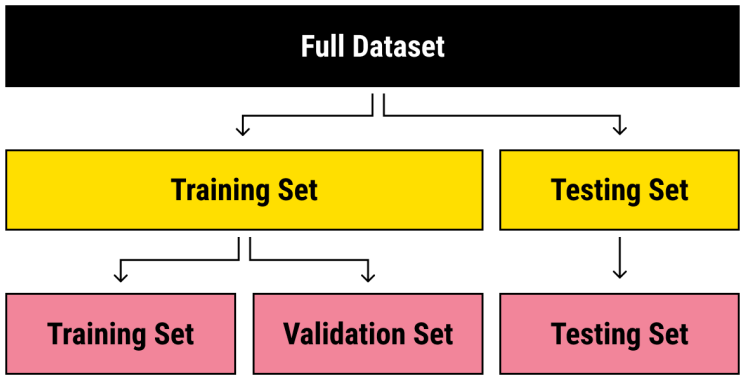
© 2026 Fessler

- Partition the **training data** as

$$\underbrace{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)}_{\text{used to fit the model } \hat{f}_\lambda}$$

$$\underbrace{(\boldsymbol{x}_{m+1}, y_{m+1}), \ldots, (\boldsymbol{x}_n, y_n)}_{\text{used to estimate } R(\hat{f}_\lambda)},$$

  perhaps after randomly shuffling the training data.
- $1 \ll m < n$
- The second part is the **validation dataset**, aka



- Use the training set for model
- Use the validation set for model

Figure courtesy [5]

Training process with validation data

© 2026 Fessler

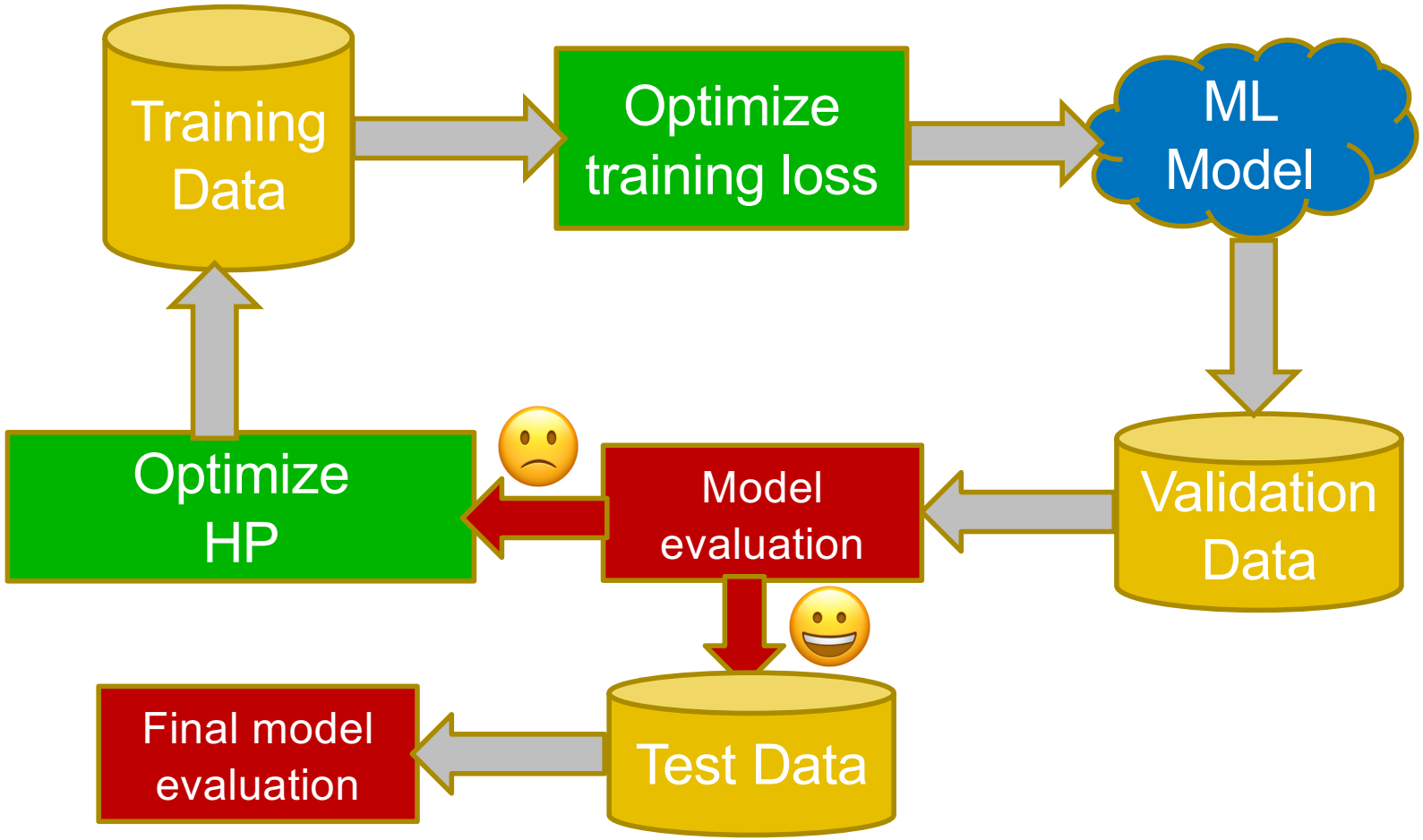


Figure courtesy Samet Oymak

The **holdout risk estimate** is

$$\hat{R}_{\mathrm{HO}}(\hat{f}_{\boldsymbol{\lambda}}) =$$

We can avoid overfitting by selecting $\boldsymbol{\lambda}$ by solving:

$$\boldsymbol{\lambda}_{\mathrm{HO}} \triangleq \arg\min_{\boldsymbol{\lambda}} \hat{R}_{\mathrm{HO}}(\hat{f}_{\boldsymbol{\lambda}}).$$

The holdout risk estimate is an **unbiased estimate** of risk:

$$\mathbb{E}_{\mathcal{D}_{\mathrm{T}}, \mathcal{D}_{\mathrm{H}}} \left\{ \hat{R}_{\mathrm{HO}}(\hat{f}_{\boldsymbol{\lambda}}) \right\} =$$

where $\mathcal{D}_{\mathrm{T}} \triangleq ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m))$ and
$\mathcal{D}_{\mathrm{H}} \triangleq ((\boldsymbol{x}_{m+1}, y_{m+1}), \ldots, (\boldsymbol{x}_n, y_n))$.

Holdout unbiased risk estimate [Read]

© 2026 Fessler

Let $\mathsf{P}\{d\}$ enote the joint distribution of $(\boldsymbol{X}, Y)$ and suppose

$$(\boldsymbol{X}_1, Y_1), \dots, (\boldsymbol{X}_n, Y_n) \overset{iid}{\sim} P.$$

Then for $i = m+1 \dots, n$: $\mathsf{p}((\boldsymbol{X}_i, Y_i) \mid \mathcal{D}_\mathsf{T}) = \mathsf{p}((\boldsymbol{X}_i, Y_i)) = P$ by IID. Thus

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}_\mathsf{H} | \mathcal{D}_\mathsf{T}} \left\{ \hat{R}_{\mathsf{HO}}(\hat{f}_\lambda) \right\} &= \frac{1}{n-m} \sum_{i=m+1}^{n} \mathbb{E}_{(\boldsymbol{X}_i, Y_i) | \mathcal{D}_\mathsf{T}} \left\{ L(Y_i, \hat{f}_\lambda(\boldsymbol{X}_i)) \right\} \\
&= \frac{1}{n-m} \sum_{i=m+1}^{n} R(\hat{f}_\lambda) \quad \text{because } \hat{f}_\lambda \text{ is fixed given } \mathcal{D}_\mathsf{T} \\
&= R(\hat{f}_\lambda).
\end{aligned}$$

The claim now follows by the **law of total expectation**:

$$\mathbb{E}_{\mathcal{D}_\mathsf{T}, \mathcal{D}_\mathsf{H}} \left\{ \hat{R}_{\mathsf{HO}}(\hat{f}_\lambda) \right\} = \mathbb{E}_{\mathcal{D}_\mathsf{T}, \mathcal{D}_\mathsf{H}} \left\{ \mathbb{E}_{\mathcal{D}_\mathsf{H} | \mathcal{D}_\mathsf{T}} \left\{ \hat{R}_{\mathsf{HO}}(\hat{f}_\lambda) \right\} \right\}. \quad \square$$

Hyperparameter selection strategies: Overview

© 2026 Fessler

Different methods for selecting $\boldsymbol{\lambda}$ using $\hat{R}_{\text{HO}}$

- Analyzing $\hat{R}_{\text{HO}}$ theoretically (rare)
- Grid search
  Example. For SVM one might try
  $(C, \sigma) \in \{1, 10, 100, 1000\} \times \{10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}\}$.
- Random search
-

Other strategies

- Common knowledge
- Tuning by hand
- Variational methods [6, p. 486, Fig. 10.9]

Hyperparameter selection: Analytically

© 2026 Fessler

- Analyzing $\hat{R}_{\text{HO}}$ theoretically
  - *e.g.*, step size is often driven by
  - or smoothness properties / properties of Hessian
  - drawback: needs information about your optimization landscape
  - drawback: often provides worst case guarantees when
- Common knowledge
- Tuning by hand
- Grid Search
- Random search
- Bayesian optimization

Hyperparameter selection: Folklore

© 2026 Fessler

- Analyzing $\hat{R}_{\mathrm{HO}}$ theoretically
- Common knowledge
  - many ML researchers might "know" what works and what does not!
  - *e.g.*, typical Adam learning rates are 10^{-4} to 10^{-3}
  - *e.g.*, typical dropout rates are $0.5 - 0.9$
  - drawback: the huge variety of ML problems often means we should not limit ourselves to what worked before
- Tuning by hand
- Grid search
- Random search
- Bayesian optimization

Hyperparameter selection: GSD

© 2026 Fessler

- Analyzing $\hat{R}_{\text{HO}}$ theoretically
- Common knowledge
- Tuning by hand (aka graduate student descent GSD)
  - Fiddle with the hyperparams until you get results you want
  - (unfortunately) very commonly used!
  - drawback: lots of effort usually required
  - drawback: not principled / generalizable / reproducible
  - tempting to cheat by looking at test data
  - but can contribute to "common knowledge"
- Grid search
- Random search
- Bayesian optimization

Hyperparameter selection: Grid search

© 2026 Fessler

- Analyzing $\hat{R}_{\text{HO}}$ theoretically
- Common knowledge
- Tuning by hand
- **Grid search** or **parameter sweep**
  - Define a range of parameters to try
  - *e.g.*, step size from 10^{-4} to 10^{-2} and $\lambda = 0.5 - 2$
  - then create a grid with several options between the endpoints
  - essentially brute force and you must choose the grid properly
  - can be viable for 1-3 dimensional hyperparameters
  - drawback: as the number of hyperparams increases, the cost of grid search increases exponentially.
- Random search
- Bayesian optimization

CQ: Grid search

© 2026 Fessler

Q16.6 Here is an example risk contour plot for hyperparameter grid search. What should the axes labels be?

A: x, y B: x_1, x_2 C: y_1, y_2 D: θ_1, θ_2 E: λ_1, λ_2

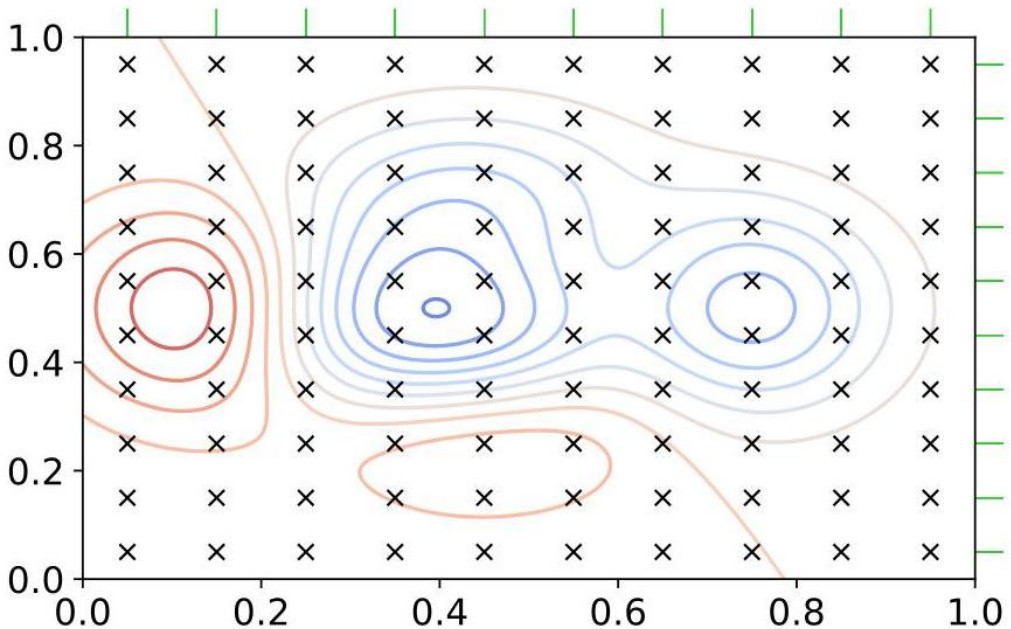


Image courtesy A. Elevers

Accelerating grid search

© 2026 Fessler

- One can speed up grid search using early stopping and successive halving
- Run all grid points for one epoch, discard the worst half and continue
- Benefit: highly parallel
- Drawback:
- Alternative: coarse to fine grid search

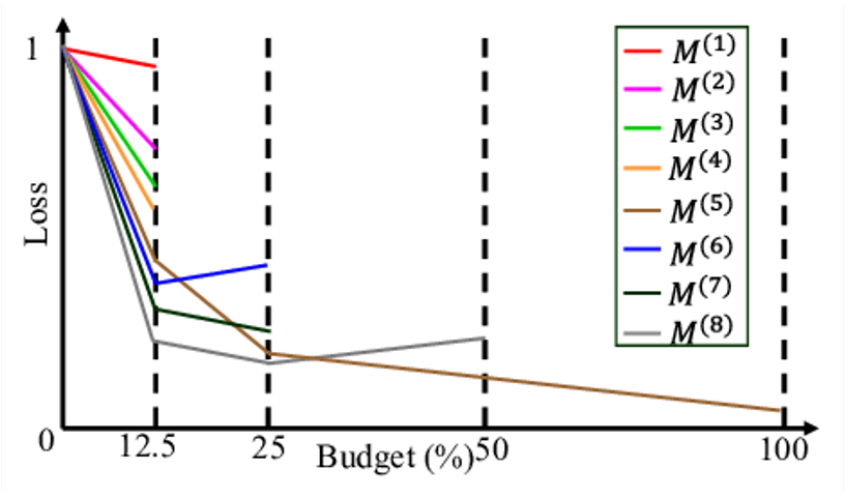


Figure courtesy A. Krunz

- Analyzing $\hat{R}_{\text{HO}}$ theoretically
- Common knowledge
- Tuning by hand
- Grid search
- **Random search**
  - Select parameter vectors randomly from the defined range
  - try however many that you can afford computationally
  - benefit: simple
  - drawback: unlikely to get close to optimal in a high-dimensional space
- Bayesian optimization

- Upper level: minimize holdout risk estimate:

$$\boldsymbol{\lambda}_{\text{HO}} \triangleq \arg\min_{\boldsymbol{\lambda}} \hat{R}_{\text{HO}}\left(\hat{f}_{\hat{\boldsymbol{\theta}}(\boldsymbol{\lambda})}\right), \quad \hat{R}_{\text{HO}}(\hat{f}) = \frac{1}{n-m} \sum_{i=m+1}^{n} L(y_i, \hat{f}(\boldsymbol{x}_i))$$

- Lower level: optimize model parameters (for given $\boldsymbol{\lambda}$):

$$\hat{\boldsymbol{\theta}}(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^p} J(\boldsymbol{\theta}; \boldsymbol{\lambda}).$$

  For example:

$$J(\boldsymbol{\theta}; \boldsymbol{\lambda}) = \frac{1}{m} \sum_{i=1}^{m} L\left(y_i, \hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right) + \lambda \left\|\boldsymbol{\theta}\right\|_2^2 .$$

- Earlier we considered $\boldsymbol{\lambda}$ to be "given" and focused on model fitting: $\arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
- Now we write $J(\boldsymbol{\theta}; \boldsymbol{\lambda})$ and $\hat{\boldsymbol{\theta}}(\boldsymbol{\lambda})$ to remind us that $\boldsymbol{\lambda}$ matters.
- Both upper level and lower level are often expensive!

Bayesian optimization: Overview

© 2026 Fessler

- **Bayesian optimization**: a sequential statistical approach for minimizing noisy black-box functions like $F(\boldsymbol{\lambda}) = \hat{R}_{\text{HO}}\left(\hat{f}_{\hat{\boldsymbol{\theta}}(\boldsymbol{\lambda})}\right)$
- **Key idea**: learn a statistical model of the function that maps hyperparameter values to the loss function
  - *e.g.*, **Gaussian Process** model
- Choose parameters to minimize the loss under this model
- **Main benefit:**
- empirically it has been demonstrated to get better results in fewer experiments
- Somewhat widely used, with lots of opportunity for innovation/research.

Bayesian optimization: Basics

© 2026 Fessler

- It deserves its **whole class**. Fundamental topic...

- Practical approach:
  - Approximate the target function F with **Gaussian Processes**
  - Future HP trials depend on the past
  - Past trials are used to
    - determine the HP choices with **good accuracy**
    - quantify the **uncertainty** of the HP choices

Start of slides from Prof. Samet Oymak, who in turn got them from someone else

Bayesian optimization: Exploration

© 2026 Fessler

- Past trials are used to
  - determine the HP choices with good accuracy
  - quantify the uncertainty of the HP choices

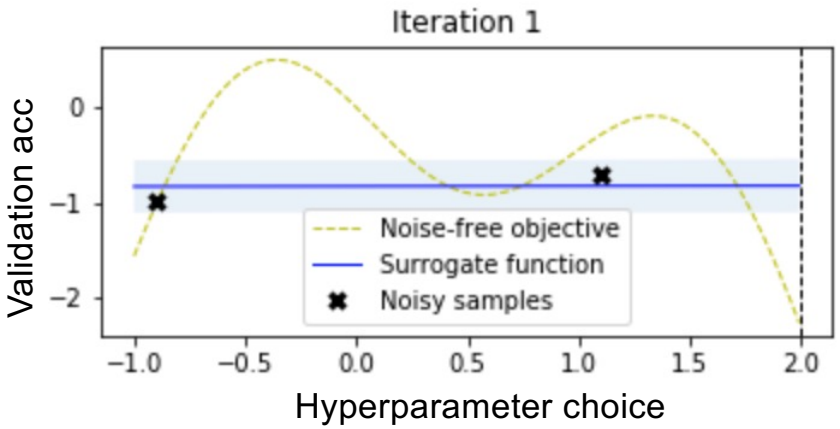
- **Exploration-Exploitation tradeoff**
  - New sample should identify promising HP choices
  - Uncertain regions might appear to have low accuracy but they might contain the optimal HP choice!

- **Acquisition function:** Optimizes exploration-exploitation tradeoff of the next sample given the history

Bayesian optimization: Iterations

© 2026 Fessler

Start the process by getting a few random samples



Q: How do we pick the next sample?

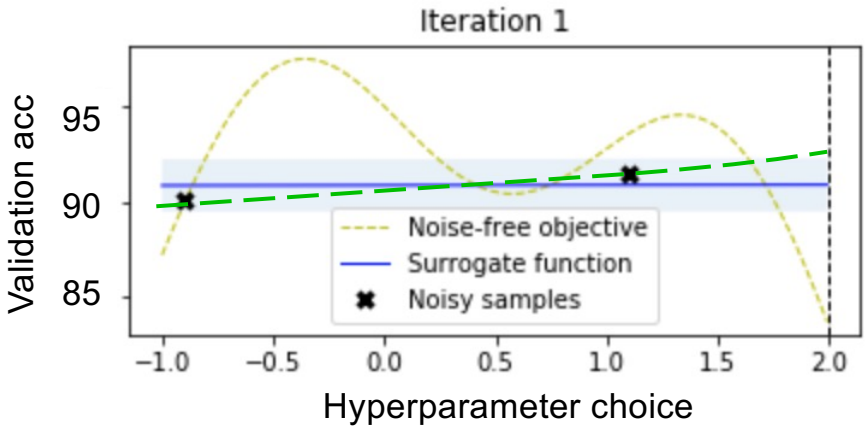
A: Use an acquisition function balancing exploration/exploitation

Surrogate function (blue): The mean of the Gaussian process
Shaded region: Will quantify the uncertainty of the target function
Yellow dashed line: The original function we want to **maximize**
Samples are allowed to be noisy i.e. instead of F(x), we see F(x)+noise

Bayesian optimization: Surrogate

© 2026 Fessler

Start the process by getting a few random samples

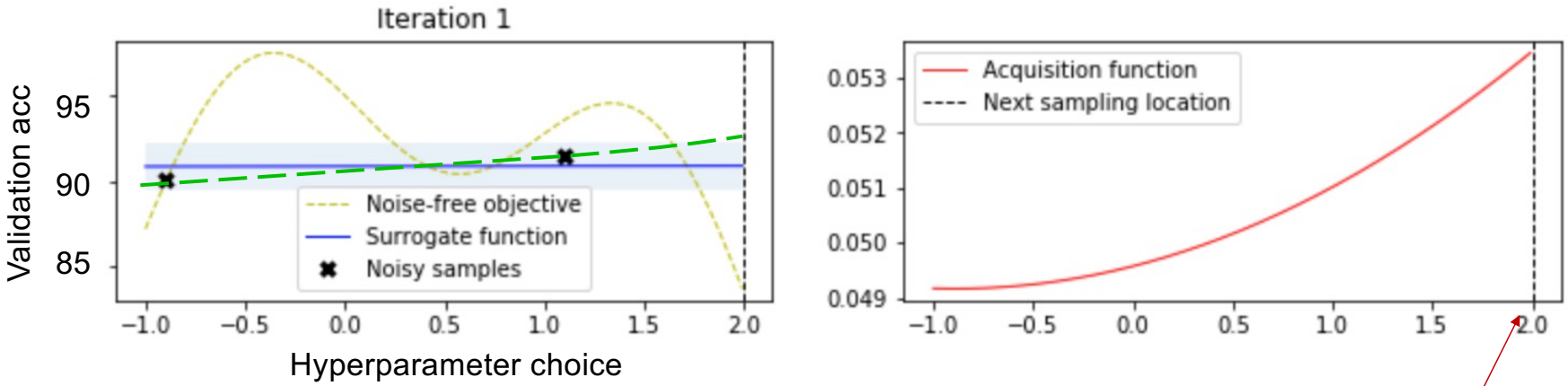


Initial surrogate function (blue): The mean of the Gaussian process
Shaded region: Will quantify the uncertainty of the target function
Yellow dashed line: The original function we want to **maximize**
Samples are allowed to be noisy i.e. instead of F(x), we see F(x)+noise
Updated surrogate after initial sampling

Bayesian optimization: Acquisition

© 2026 Fessler

Start the process by getting a few random samples

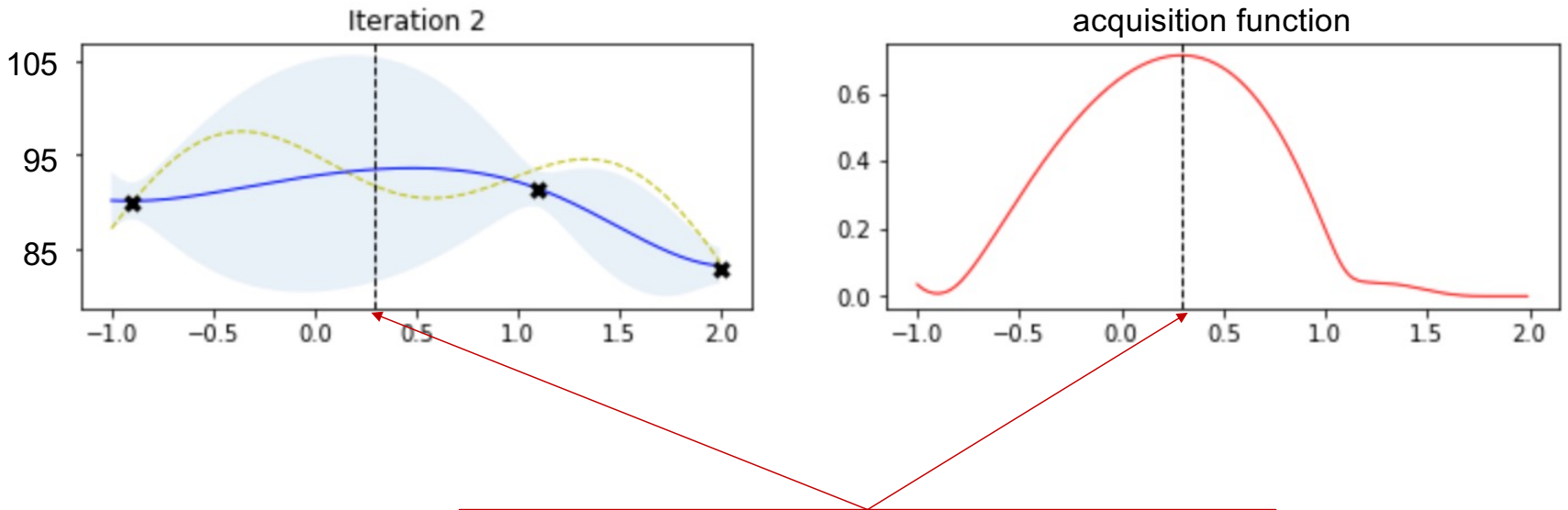


Initial surrogate function (blue): The mean of the Gaussian process
Shaded region: Will quantify the uncertainty of the target function
Yellow dashed line: The original function we want to **maximize**
Samples are allowed to be noisy i.e. instead of F(x), we see F(x)+noise
Updated surrogate after initial sampling

Location of the next sample

Bayesian optimization: Update 2

© 2026 Fessler

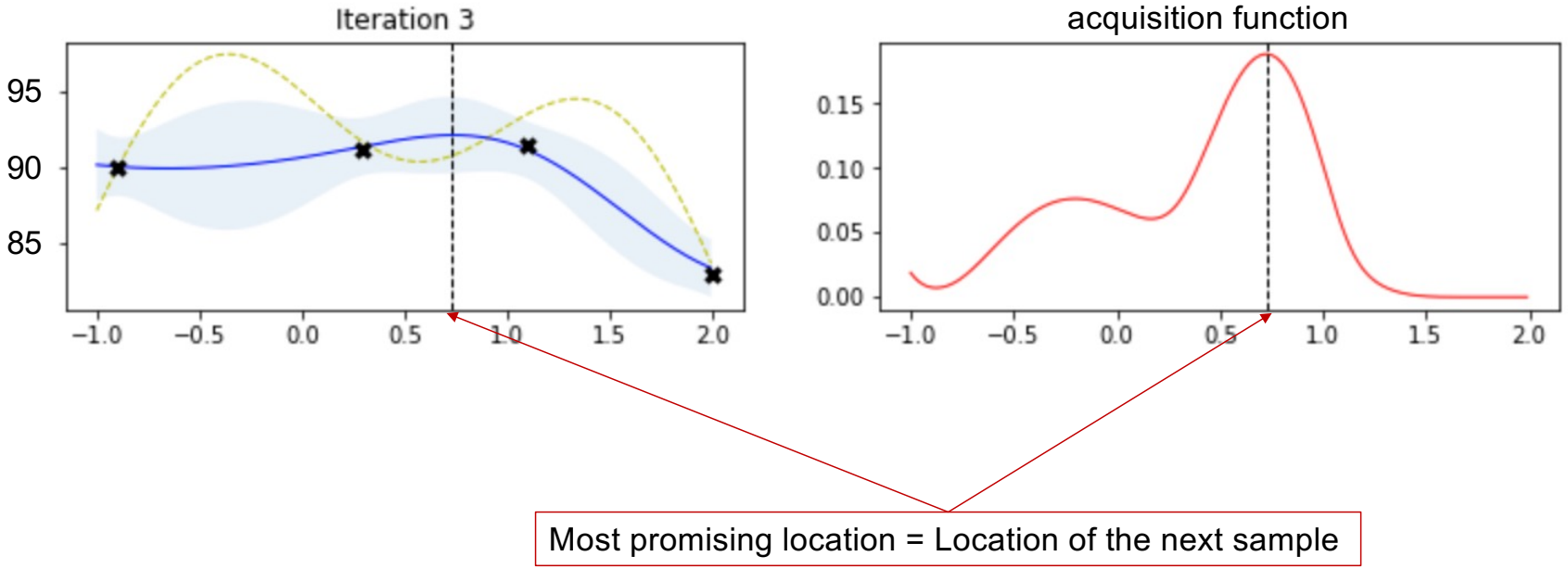


Most promising location = Location of the next sample

Exploration = Sample regions with high uncertainty
Exploitation = Sample regions that has high expected reward

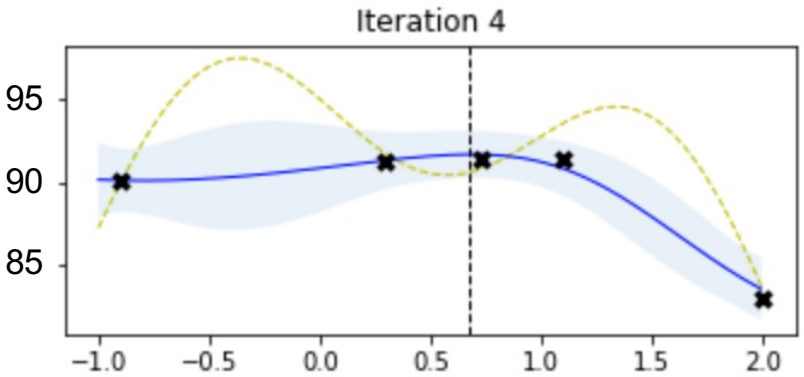
Bayesian optimization: Update 3

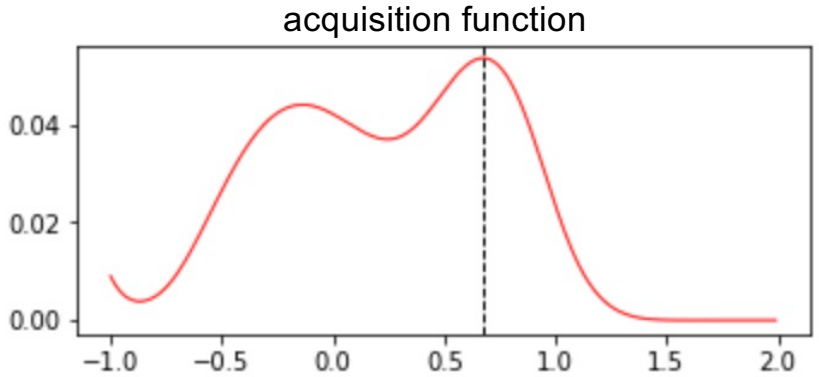
© 2026 Fessler



Bayesian optimization: Update 4

© 2026 Fessler

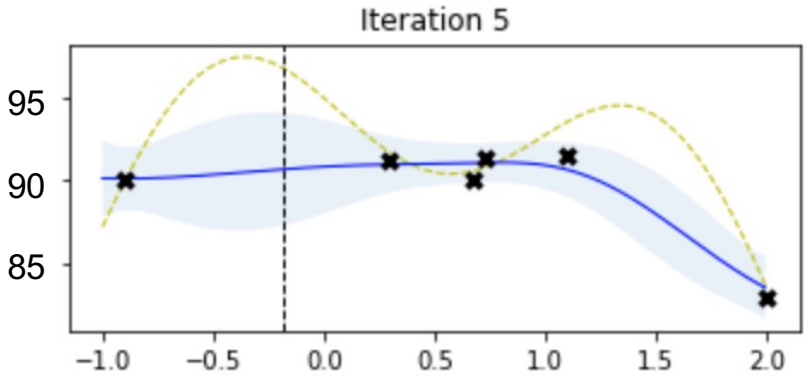


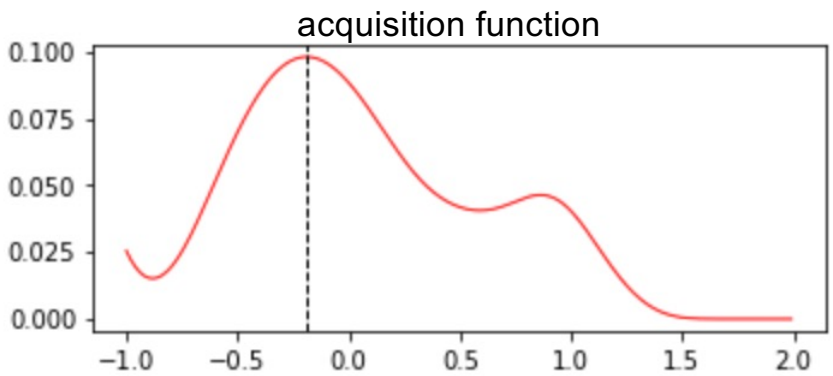


Surrogate function (blue): The mean of the Gaussian process
Shaded region: Will quantify the uncertainty of the target function
Yellow dashed line: The original function we want to **maximize**
Samples are allowed to be noisy i.e. instead of F(x), we see F(x)+noise

Bayesian optimization: Update 5

© 2026 Fessler

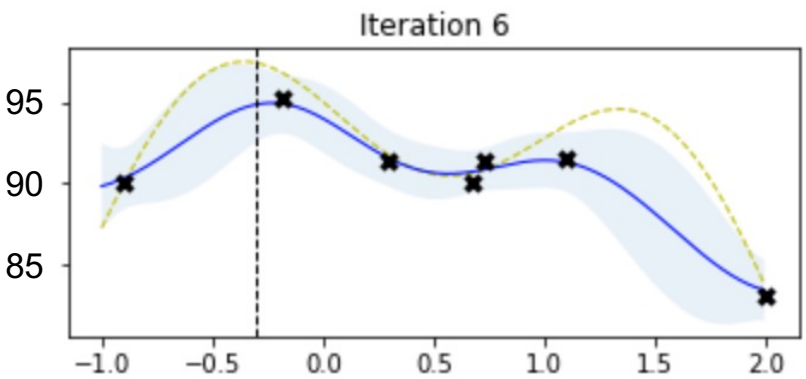


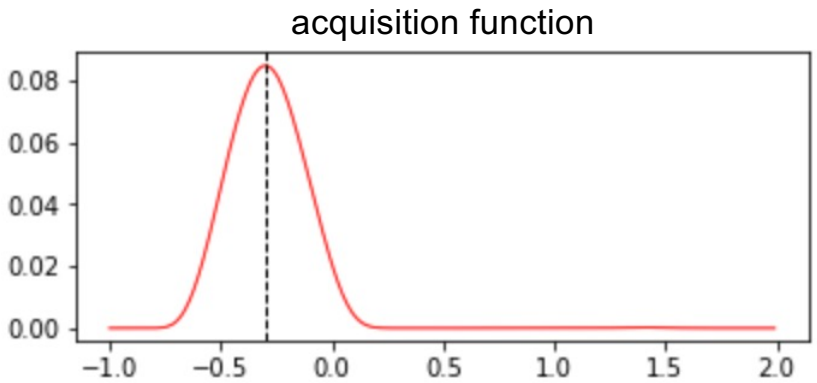


Remark: As we have more samples, the uncertainty is decreasing!

Bayesian optimization: Update 6

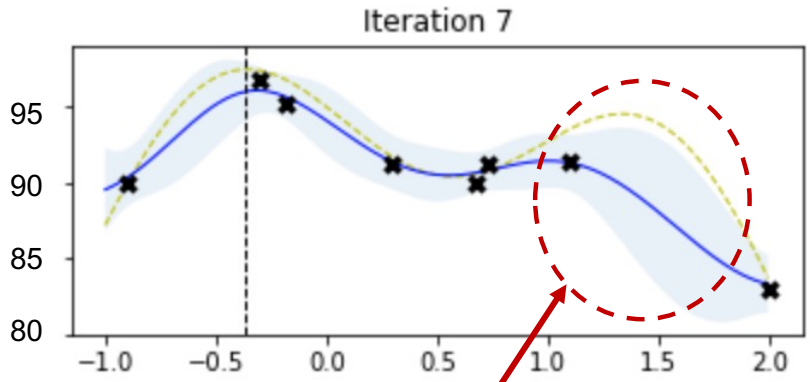
© 2026 Fessler

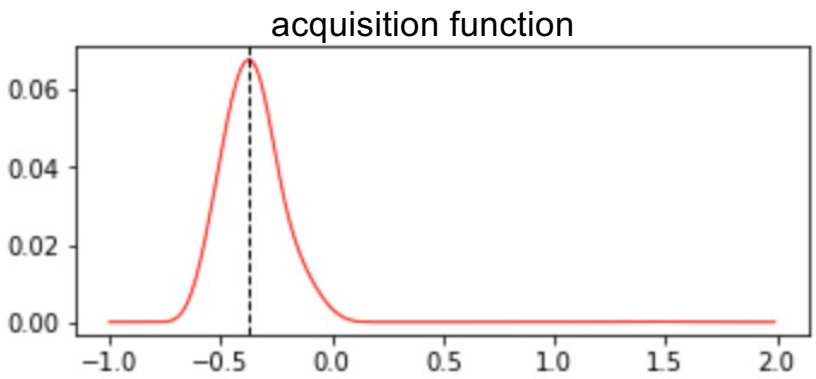




Bayesian optimization: Update 7

© 2026 Fessler

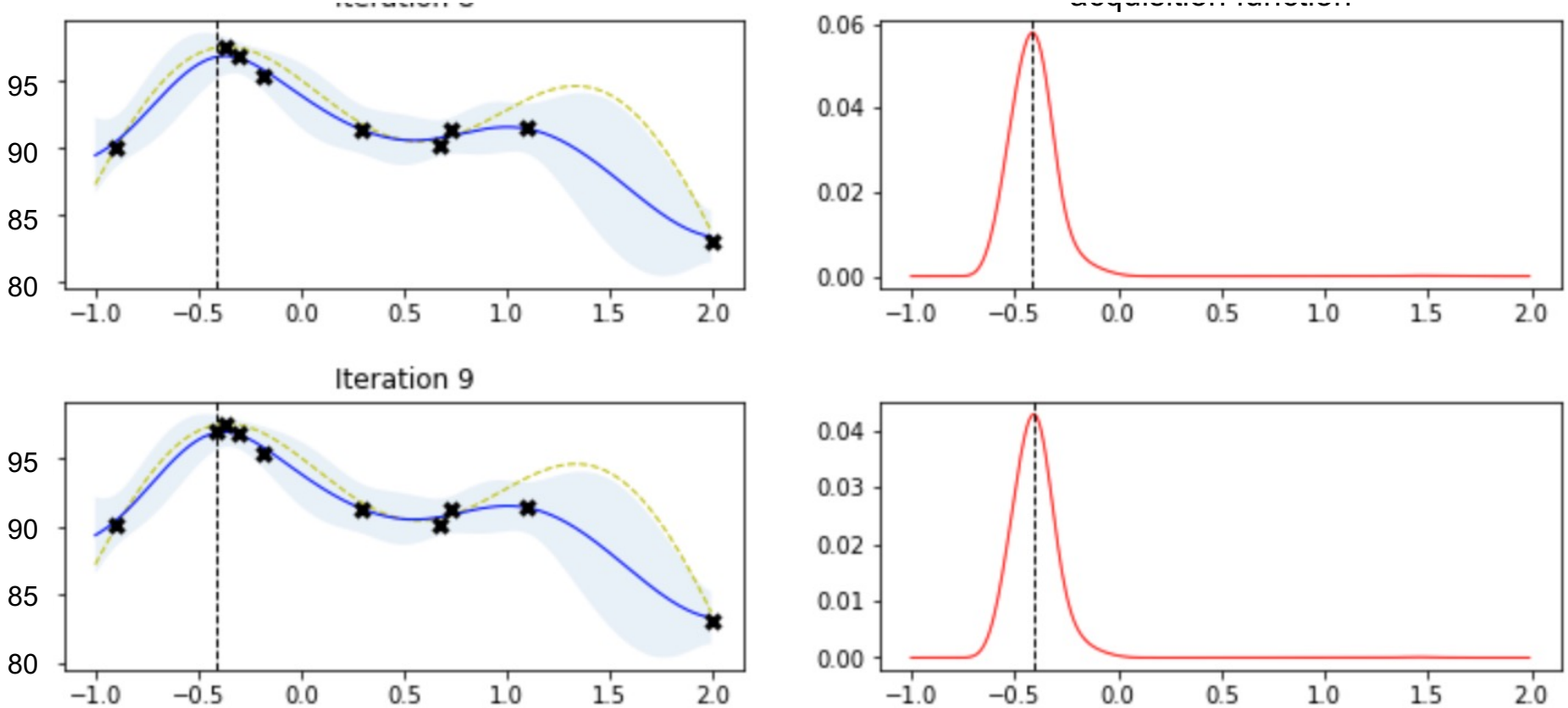




No need to explore here.
Uncertainty is not **strong** enough!

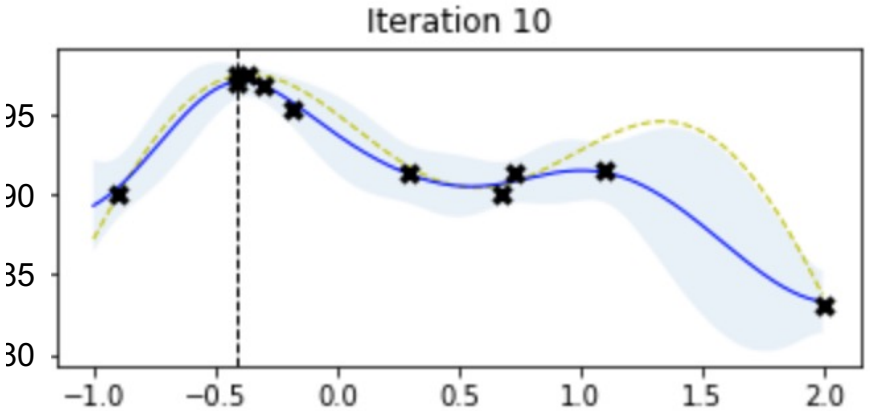
Bayesian optimization: Update 8-9

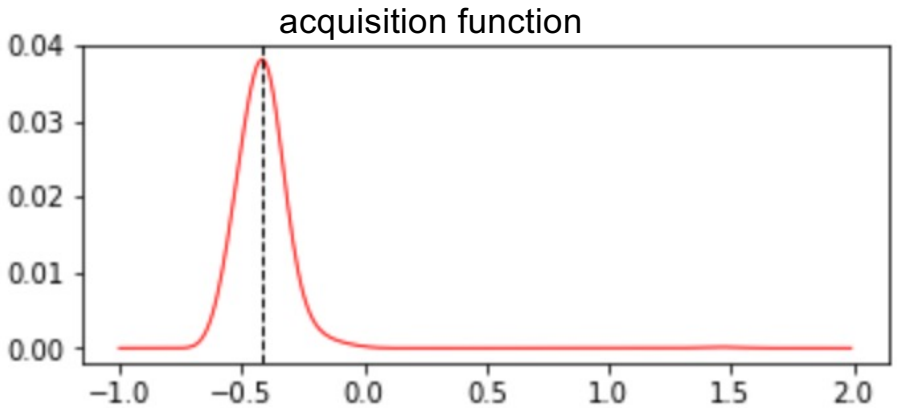
© 2026 Fessler



Bayesian optimization: Update 10

© 2026 Fessler







We can STOP!

References I

[1] K. P. Murphy.
Probabilistic machine learning: An introduction.
MIT Press, 2022.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
Advances in neural information processing systems, 30, 2017.

[3] O. Hsu and K. Koul.
Better hardware could turn zeros into AI heroes: Sparse computing enables leaner, faster AI.
IEEE Spectrum, 63(6):28–33, 2026.

[4] Jacob Austin, Sholto Douglas, Roy Frostig, Anselm Levskaya, Charlie Chen, Sharad Vikram, Federico Lebron, Peter Choy, Vinay Ramasesh, Albert Webson, and Reiner Pope.
How to Scale Your Model.
Google DeepMind, 2025.
Retrieved from https://jax-ml.github.io/scaling-book/.

[5] Iryna Sydorenko, 2021.
https://labelyourdata.com/articles/machine-learning-and-training-data.

[6] C. Bishop.
Pattern recognition and machine learning.
Springer, 2006.

[7] scikit learn.
3.1. cross-validation: evaluating estimator performance, 2011.
https://scikit-learn.org/stable/modules/cross_validation.html.