Welcome!

DOMINO

Getting Started with Hyperparameter Optimisation

Nikolay Manchev
Head of Data Science for EMEA
Domino Data Lab

Workshop Do's and Don'ts

- Learn about generalisation, GP, and hyperparameter tuning
- It will quickly get complicated
- There will be lots of hands-on (JupyterLab, Python, Ray)
- You'll get access to the slides & notebooks
- You can keep your Domino access

The Domino Enterprise MLOps Platform

A force multiplier for data science in the enterprise

Data: S3, hadoop HDFS, SQL, CSV, snowflake

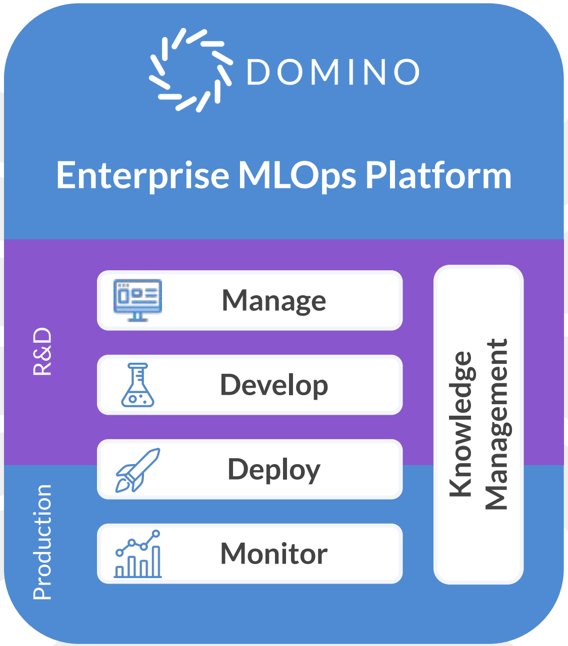
Software: GitHub, JIRA, Amazon SageMaker

Languages: python, R, sas, MATLAB

Tools: jupyter, RStudio, Apache Zeppelin, ANACONDA

Packages: Shiny, PyTorch, scikit learn, H2O.ai, DataRobot

Compute: Apache Spark, RAY, DASK, NVIDIA



Infrastructure: aws

Open & Flexible
Use the tools & infrastructure you want

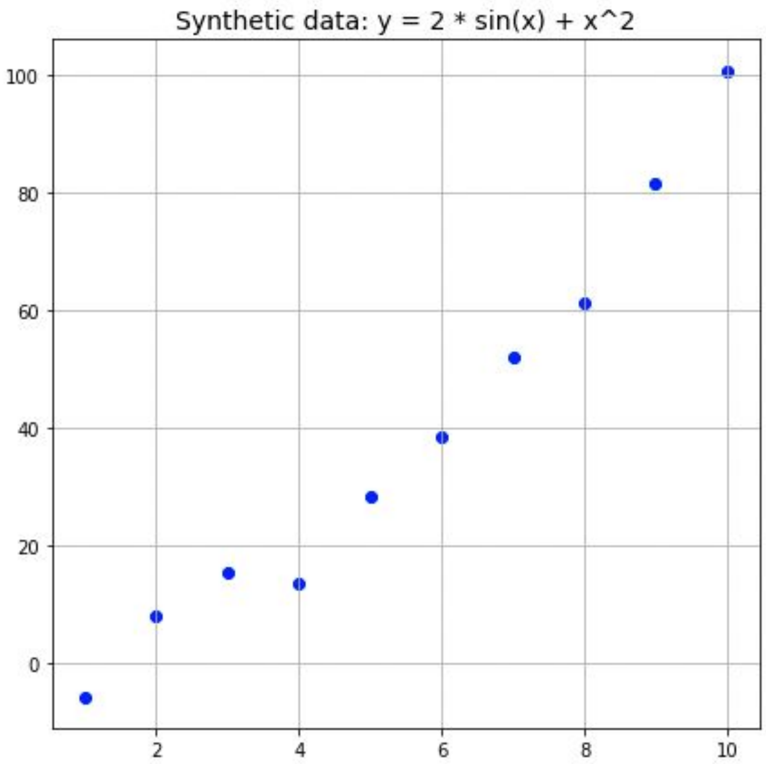
Built for Teams
Reproduce work and compound knowledge

Integrated Workflows
Reduce friction across the end-to-end lifecycle

Enterprise Scale
Safely and universally scale data science

DOMINO

Example



Curve fitting

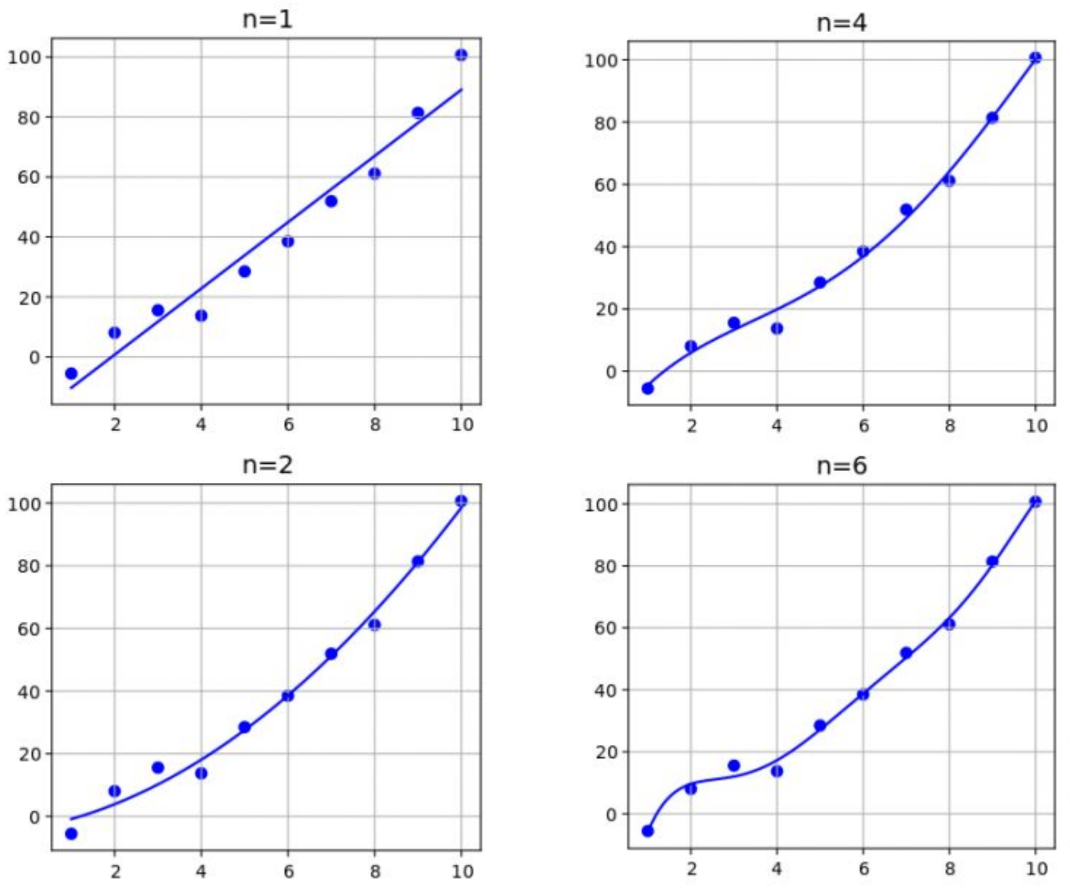
Polynomial regression

$$y = \beta_0 + \beta_1 x^1 + \beta_2 x^2 + \cdots + \beta_n x^n$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ 1 & x_3 & x_3^2 & \cdots & x_3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

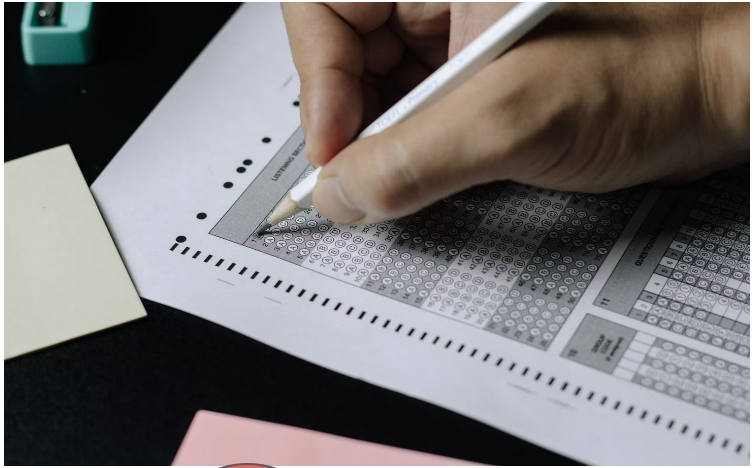
$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}$$

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$



Generalisation

The capability of our model to adapt to previously unseen data



Teacher: "To reduce a fraction, divide the numerator and denominator by the greatest common factor.

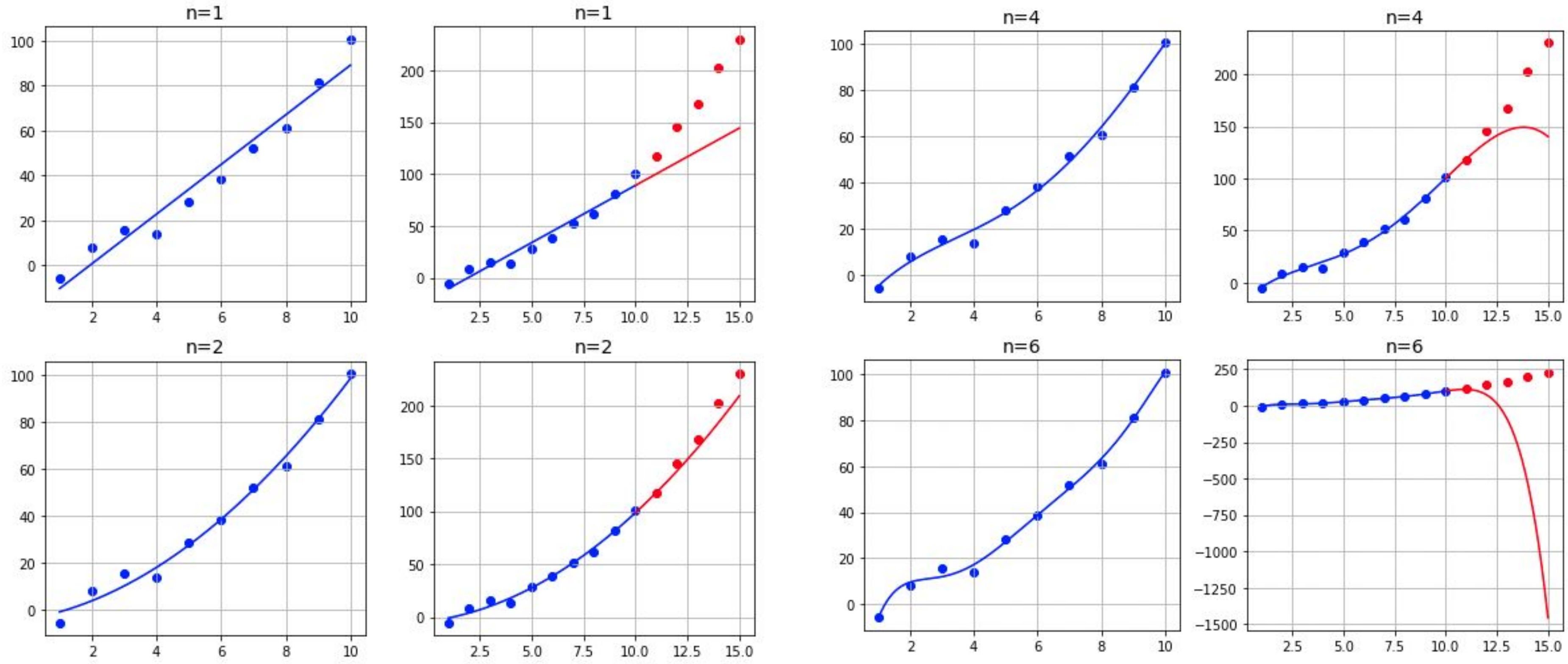
For example, 8/24 = 1/3

Exam: 8[illegible] = ?

3/12 = ?

14/49 = ?

Testing on unseen data



Cross validation

Hands-on - Domino 101

Hyperparameter search

- Find the best performing set of hyperparameters

$$\lambda^{(*)} = \underset{\lambda \in \Lambda}{\operatorname{argmin}} \; \mathbb{E}_{x \sim \mathcal{G}_x} [\mathcal{L}\left(x; \mathcal{A}_\lambda(\mathcal{X}^{(\text{train})})\right)]$$

- Gaussian kernel SVM $\lambda = (C, \gamma)$ $C \in \{10, 100, 1000\}$

$$\gamma \in \{0.1, 0.2, 0.5, 1.0\}$$

- Cross-validation

$$\lambda^{(*)} \approx \underset{\lambda \in \Lambda}{\operatorname{argmin}} \; \underset{x \in \mathcal{X}^{(\text{valid})}}{\operatorname{mean}} \; \mathcal{L}\left(x; \mathcal{A}_\lambda(\mathcal{X}^{(\text{train})})\right)$$

Naive Grid search

- Try all possible combinations from a manually specified search space

- $|\Lambda|$ grows very quickly

$$C \in \{10, 100, 1000\}$$
$$\gamma \in \{0.1, 0.2, 0.5, 1.0\}$$

$$\begin{aligned} |\Lambda| &= |C| \times |\gamma| \\ &= 3(4) = 12 \end{aligned}$$

$$C \in \{1, 10, 100, 1000\}$$
$$\gamma \in \{0.1, 0.2, 0.5, 1.0\}$$

$$\begin{aligned} |\Lambda| &= |C| \times |\gamma| \\ &= 4(4) = 16 \end{aligned}$$

$$C \in \{1, 10, 100, 1000\}$$
$$\gamma \in \{0.1, 0.2, 0.5, 1.0\}$$
$$s \in \{1, 2, 3, 4\}$$

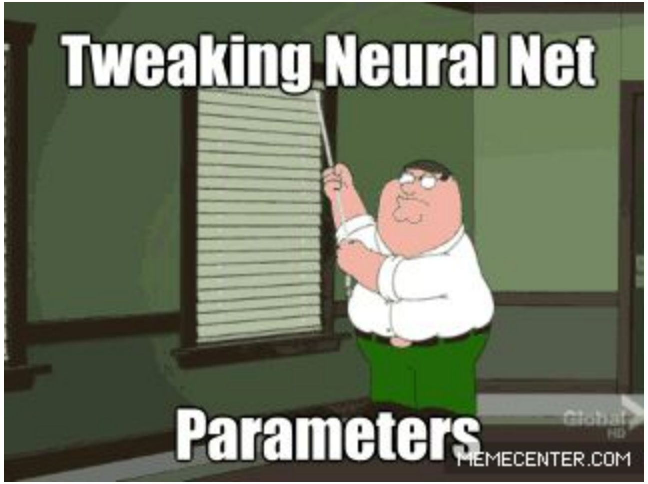
$$\begin{aligned} |\Lambda| &= |C| \times |\gamma| \times |s| \\ &= 4(4)(3) = 48 \end{aligned}$$

Grid search - pros & cons

- **Pros**
  - Easy to implement
  - Trivial to parallelise
  - Typically finds better $\lambda^{(*)}$ compared to manual search
- **Cons**
  - Very expensive
  - Greatly sufferers from Curse of Dimensionality
  - Doesn't consider past experience

Example: Optimizing NN is hard

- Number of layers
- Type of layers
- Neurons per layer
- Activation functions
- Regularizers
- Normalization
- Optimizer specific settings
    - Type
    - Learning rate(s)
    - Momentum

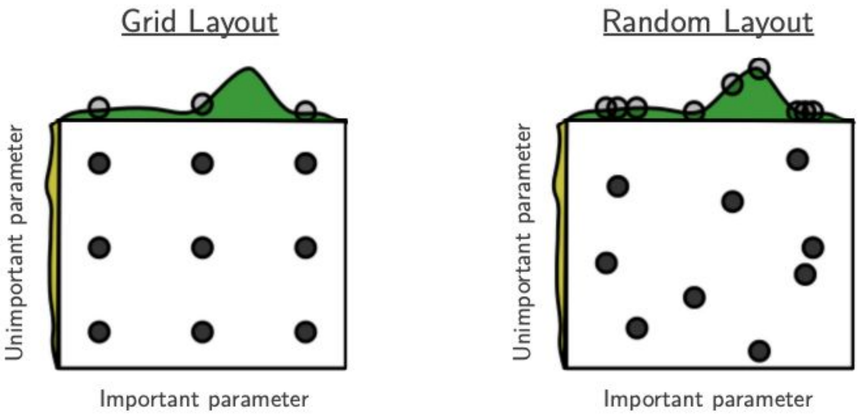


- We tend to cheat
- It may be challenging to spot patterns

Hands-on - Lab 00 - Task 1 & 2

Random Search

- Sampling random combinations
- This works well because
  - Some parameters have a small effect



J. Bergstra, and Y. Bengio., Random Search for Hyper-Parameter Optimization, Journal of Machine Learning Research 13 (Feb): 281-305 (2012)

Hands-on - Lab 00 - Task 3

LET'S PLAY A GAME

LET'S PLAY A GAME

- **We are optimising α for certain A in order to minimise L**
- **So far**

  $\alpha = 0.0001 \rightarrow L = 0.4233$

  $\alpha = 0.0002 \rightarrow L = 0.4331$

  $\alpha = 0.0004 \rightarrow L = 0.4209$

  $\alpha = 0.0004 \rightarrow L = 0.4209$

  $\alpha = 0.1000 \rightarrow L = 1.9209$

- **Each player must pick a unique α. The player with the lowest L lives.**

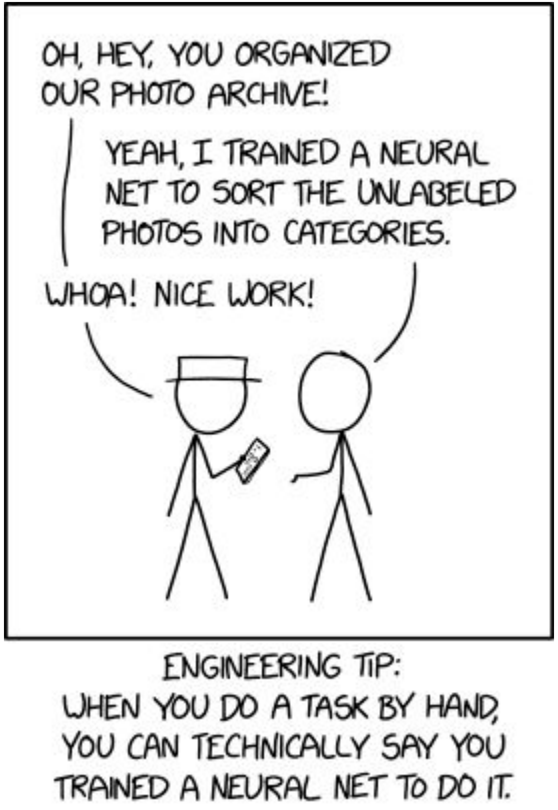
Gaussian Processes

Can we apply ML to find the hyperparameters?

- What regions of the space we think are better?
- How certain are we?

GP are ideally suited for cases where

- We don't have much prior knowledge
- Expecting similar inputs to have *similar outputs*
- Derived using probabilities



https://xkcd.com/2173/

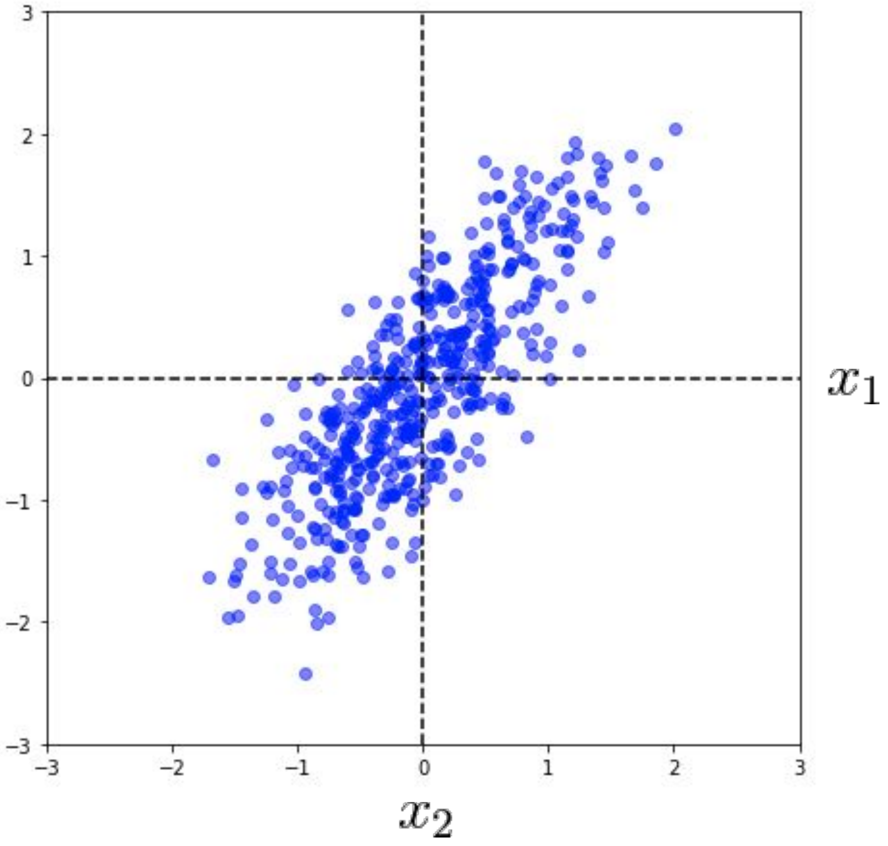
Gaussian Basics

Two variables - x_1 and x_2

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

Assuming that $\boldsymbol{x}$ follows a Gaussian

$$\boldsymbol{x} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \boldsymbol{\Sigma}\right) \equiv \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right)$$



Gaussian Basics

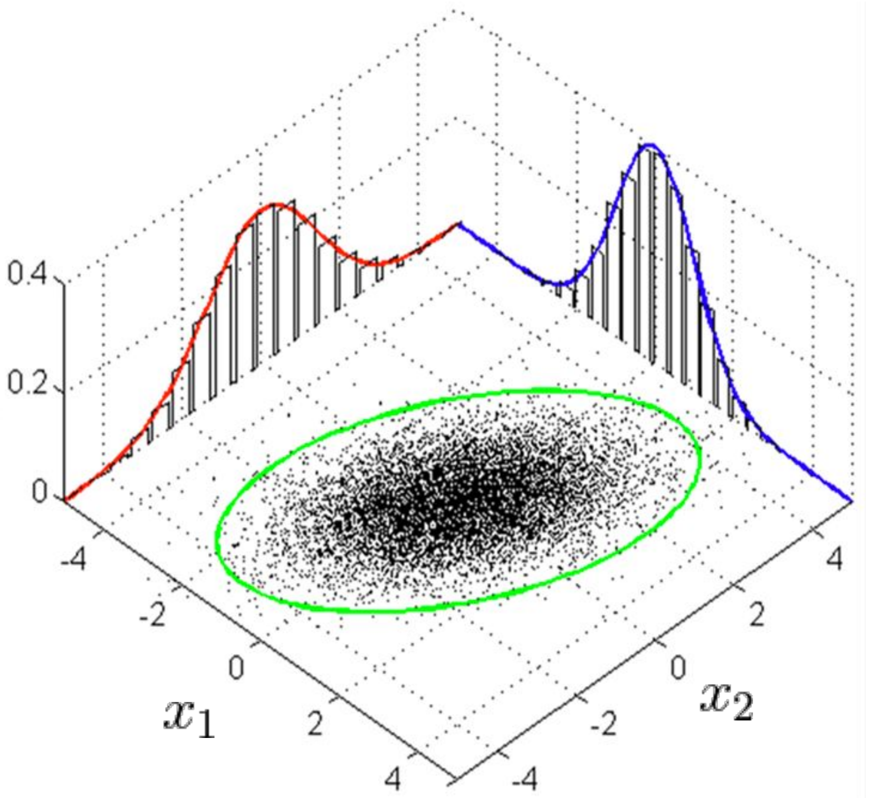
Two variables - x_1 and x_2

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

Assuming that $\boldsymbol{x}$ follows a Gaussian

$$\boldsymbol{x} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \boldsymbol{\Sigma}\right) \equiv \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right)$$

Can we find $p(x_1|x_2)$?



Bivariate case

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

The marginals are given by

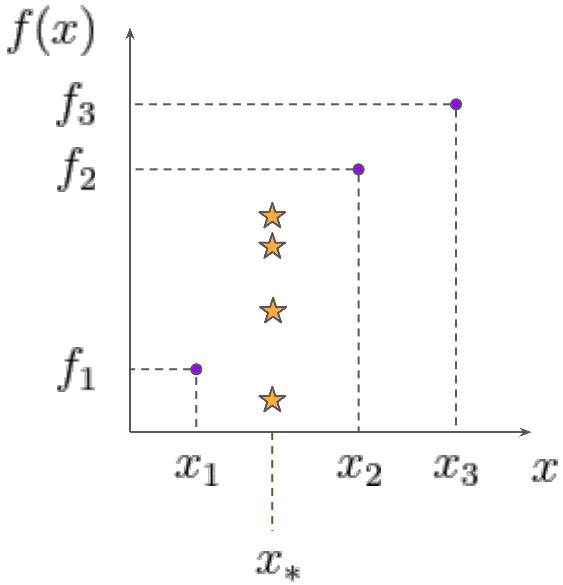
$$\begin{aligned} p(x_1) &= \mathcal{N}(x_1|\mu_1, \Sigma_{11}) \\ p(x_2) &= \mathcal{N}(x_2|\mu_2, \Sigma_{22}) \end{aligned}$$

The conditional is given by

$$\begin{aligned} p(x_1|x_2) &= \mathcal{N}(\bar{\mu}, \bar{\Sigma}) \\ \bar{\mu} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) \\ \bar{\Sigma} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \end{aligned}$$

See Conditional Distributions in Multivariate normal distribution (Wikipedia)

Estimating f(x)



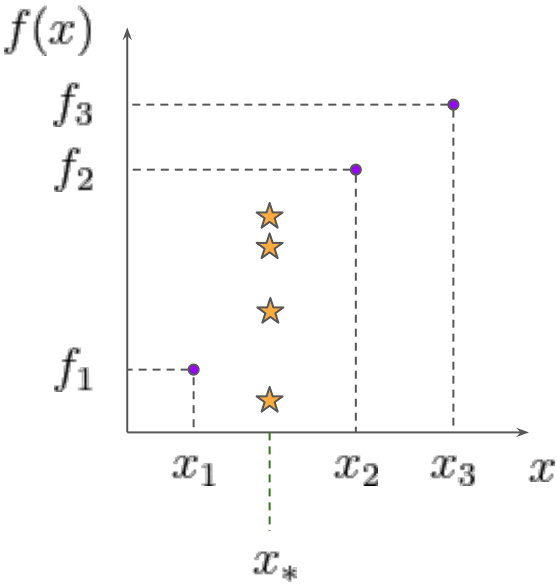
We want to estimate $f(x)$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{K}) \equiv \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{11} K_{12} K_{13} \\ K_{21} K_{22} K_{23} \\ K_{31} K_{32} K_{33} \end{bmatrix} \right)$$

$$K_{ij} = k(x_i, x_j) = e^{-||x_i - x_j||^2}$$

Find f_* given $\{(x_1, f_1), (x_2, f_2), (x_3, f_3)\}$ and x_*

Estimating f(x)



Assuming $\boldsymbol{f} \sim \mathcal{N}(0, \boldsymbol{K})$

Assuming $f_* \sim \mathcal{N}(0, K_{**}) \equiv \mathcal{N}(0, k(x_*, x_*))$

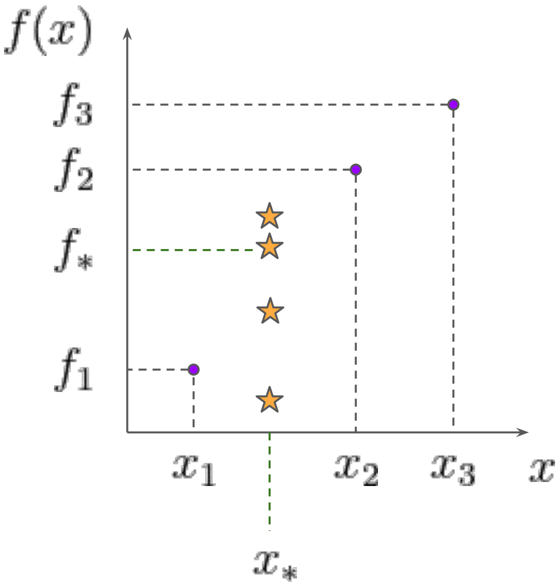
But f_* and $\boldsymbol{f}$ are independent (a.k.a. useless)

$$\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \begin{bmatrix} K_{11} K_{12} K_{13} \\ K_{21} K_{22} K_{23} \\ K_{31} K_{32} K_{33} \end{bmatrix} & \begin{bmatrix} K_{1*} \\ K_{2*} \\ K_{3*} \end{bmatrix} \\ \begin{bmatrix} K_{*1} K_{*2} K_{*3} \end{bmatrix} & K_{**} \end{bmatrix} \right) \equiv$$

$$\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{\mu}_0, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*{}^T & K_{**} \end{bmatrix} \right)$$

We made it

Estimating f(x)



Remember

Bivariate case

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

The marginals are given by

$$p(x_1) = \mathcal{N}(\mu_1, \Sigma_{11})$$
$$p(x_2) = \mathcal{N}(\mu_2, \Sigma_{22})$$

The conditional is given by

$$p(x_1|x_2) = \mathcal{N}(\bar{\mu}, \bar{\Sigma})$$
$$\bar{\mu} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$
$$\bar{\Sigma} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

See Conditional Distributions in [Multivariate normal distribution](#) (Wikipedia)

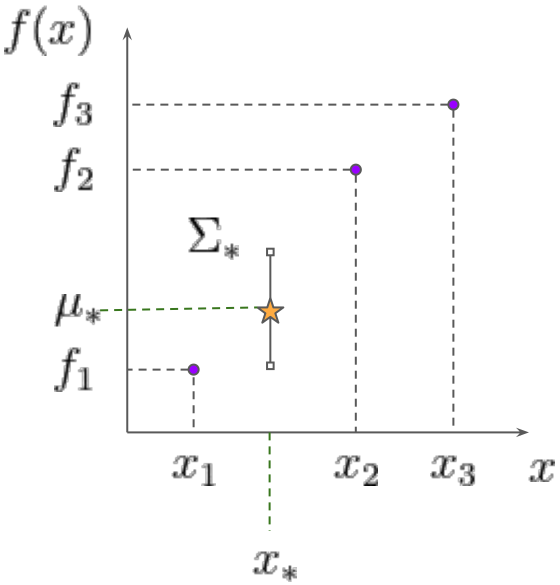
Applying this to $\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{\mu}_0, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*{}^T & K_{**} \end{bmatrix}\right)$

gives us

$$\mu_* = K_*^T K^{-1} \boldsymbol{f}$$
$$\Sigma_* = K_{**} - K_*^T K_*^{-1}$$

Yes!

Estimating f(x)



Remember

Bivariate case

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

The marginals are given by

$$p(x_1) = \mathcal{N}(\mu_1, \Sigma_{11})$$
$$p(x_2) = \mathcal{N}(\mu_2, \Sigma_{22})$$

The conditional is given by

$$p(x_1|x_2) = \mathcal{N}(\bar{\mu}, \bar{\Sigma})$$
$$\bar{\mu} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$
$$\bar{\Sigma} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

See Conditional Distributions in Multivariate normal distribution (Wikipedia)

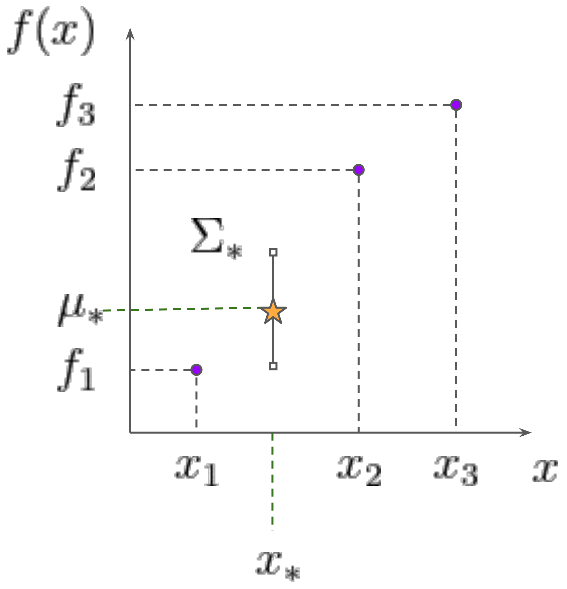
Applying this to $$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{\mu}_0, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*{}^T & K_{**} \end{bmatrix}\right)$$

gives us

$$\mu_* = K_*^T K^{-1} \boldsymbol{f}$$
$$\Sigma_* = K_{**} - K_*^T K_*^{-1}$$

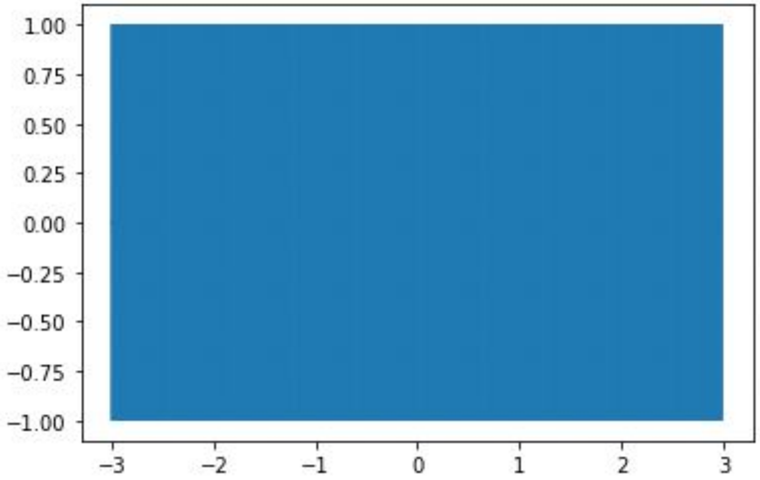
GP for Hyperparameter Optimisation

- We can use GP to approximate an objective function
- Optimisation is performed as follows:
  1. Build a surrogate model
  2. Find promising hyperparameters
  3. Test them against the real model
  4. Update the surrogate
  5. Repeat

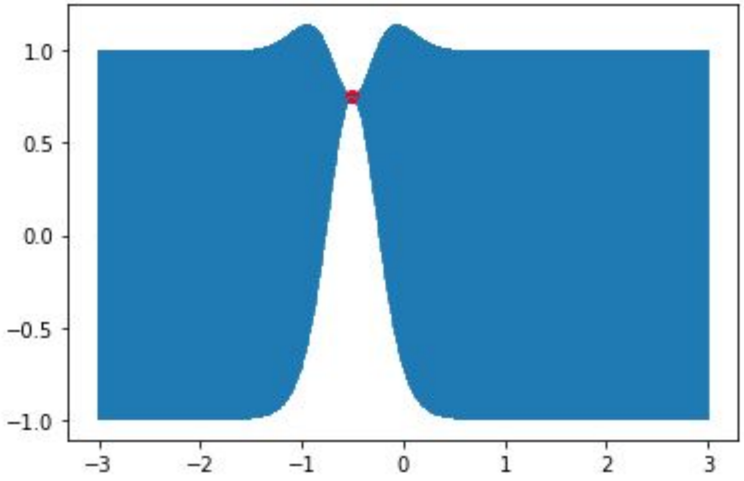


Note: There are different model choices for the approximator (GP, TPE etc.)

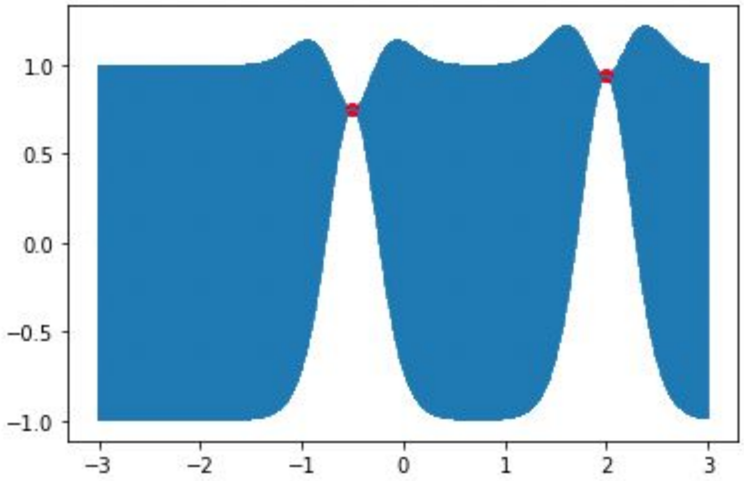
GP Example



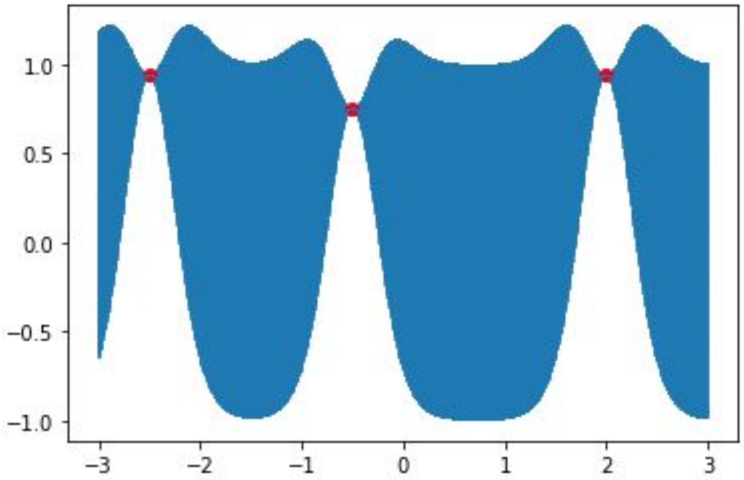
GP Example



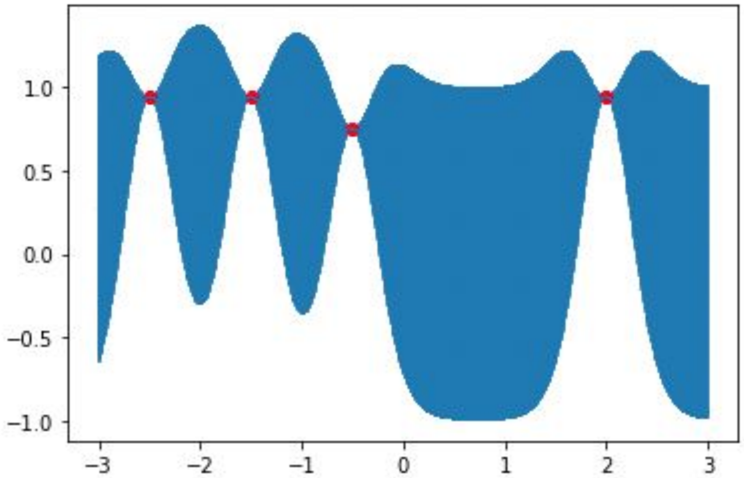
GP Example



GP Example



GP Example



Hands-on - Lab 01

HyperOpt

- Easy-to-use implementation of a Bayesian hyperparameter optimization
- Needs four pieces of information
    - objective function
    - search space
    - algorithm
    - storage

```python
from hyperopt import fmin, tpe, hp

best = fmin(fn=lambda x: x ** 2,
            space=hp.uniform('x', -10, 10),
            algo=tpe.suggest,
            max_evals=100)
print(best)
```

Bergstra, J., Yamins, D., Cox, D. D. (2013) Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. To appear in Proc. of the 30th International Conference on Machine Learning (ICML 2013).

Hands-on - Lab 03

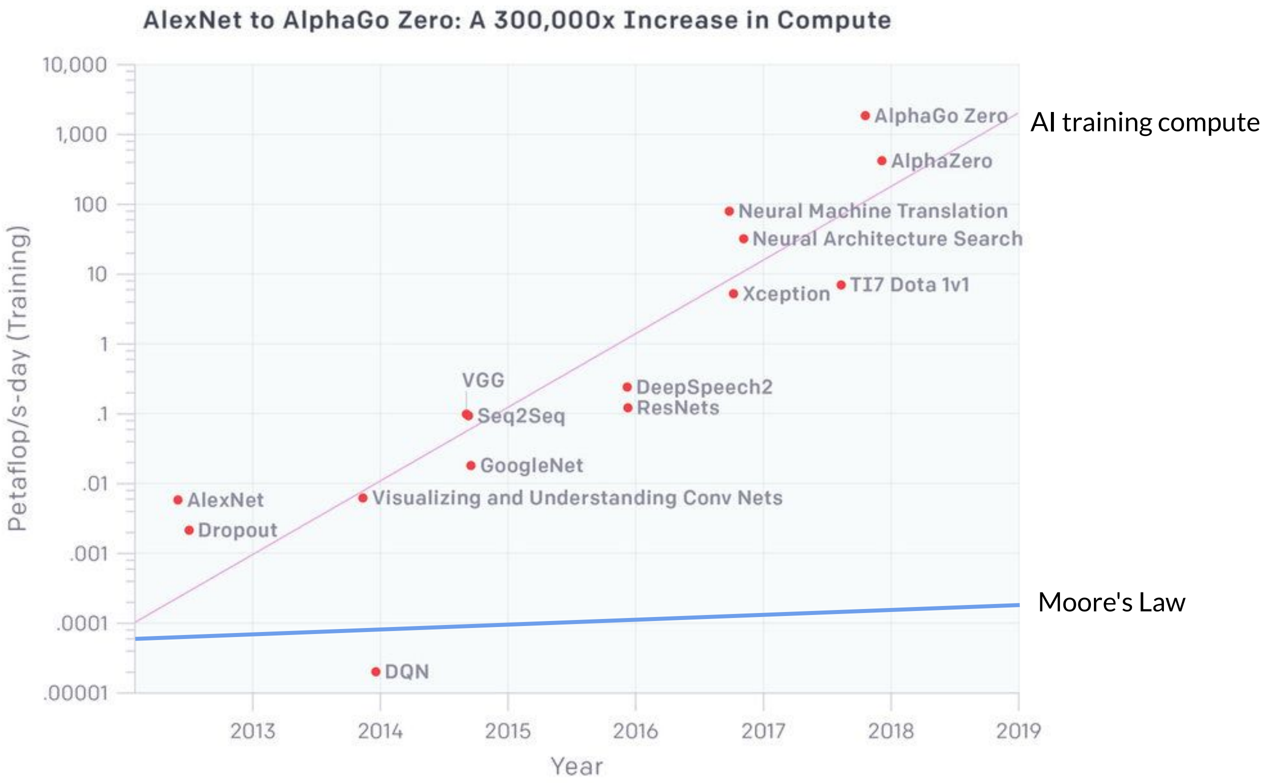
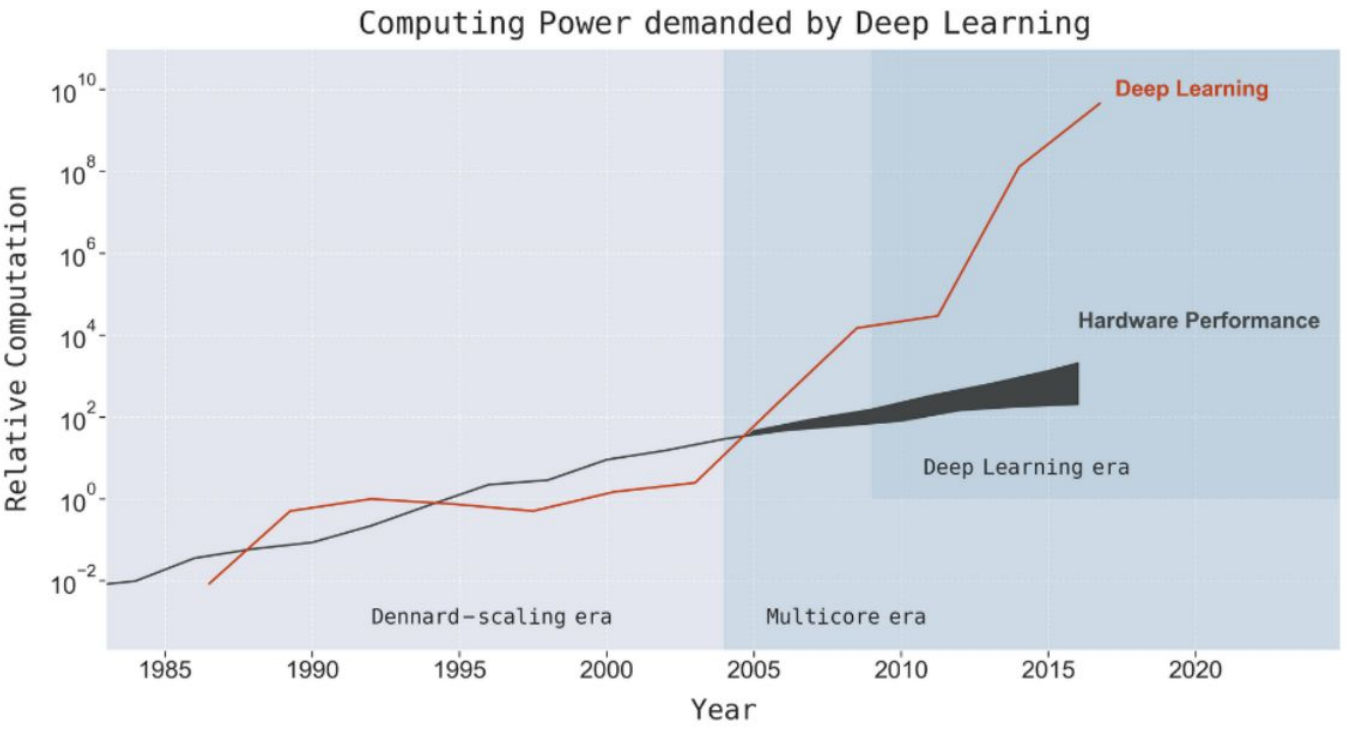


Illustration of the increasing compute demand from AlexNet in 2013 to AlphaGo Zero today; the exponential fit of the data points gave a doubling time 3.43 months, as given in Kozma, Robert & Noack, Raymond & Siegelmann, Hava. (2019). Models of Situated Intelligence Inspired by the Energy Management of Brains. 567-572. 10.1109/SMC.2019.8914064.



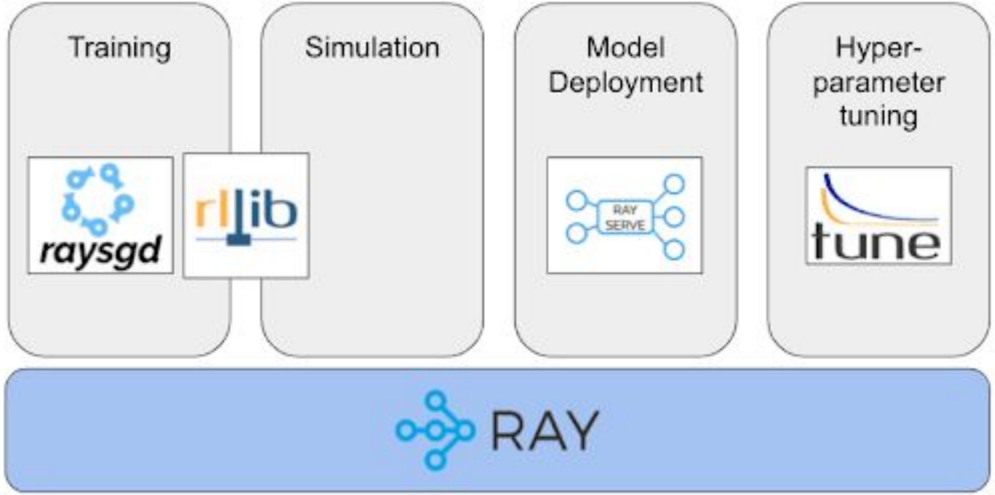
Deep learning models of all types (as compared with the growth in hardware performance from improving processors - Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz. CPU DB: **Recording microprocessor history**. Queue, 10(4):10:10–10:27, 2012.), as analyzed by a) John L. Hennessy and David A. Patterson. **Computer Architecture: A Quantitative Approach**. Morgan Kaufmann, San Francisco, CA, sixth edition, 2019 and b)] Charles E. Leiserson, Neil C. Thompson, Joel Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. **There's plenty of room at the top: What will drive growth in computer performance after Moore's law ends?** Science, 2020.

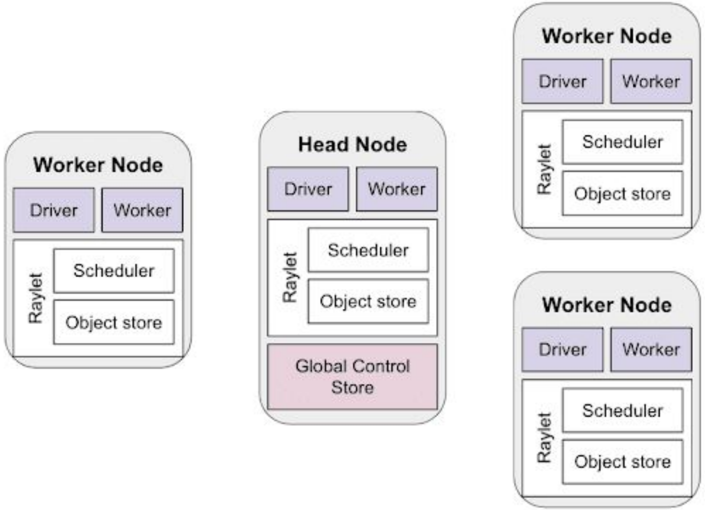
Figure from Neil C. Thompson1, Kristjan Greenewald2, Keeheon Lee3, Gabriel F. Manso, **The Computational Limits of Deep Learning**, arXiv:2007.05558v1 [cs.LG] 10 Jul 2020

DOMINO


Multithreaded programming
Theory
Actual

About Ray





- Simple, concise, and intuitive API
- Easy for people without distributed computing experience
- Flexible for a wide class of problems

"Rayifing" code

- Standard Python

```
def make_array(...):
  a = … # Construct NumPy array
  return a

def add_arrays(a,b):
  return np.add(a,b)
```

```
a = make_array(...)
b = make_array(...)
c = add_arrays(a,b)
```

Executed sequentially

- In Ray

```
@ray.remote
def make_array(...):
  a = … # Construct NumPy array
  return a

@ray.remote
def add_arrays(a,b):
  return np.add(a,b)
```

```
a = make_array.remote(...)
b = make_array.remote(...)
c = add_arrays.remote(a,b)
ray.get(c)
```

Executed asynchronously

How does it work?

When provisioning your on-demand Ray cluster, Domino sets up environment variables that hold the information easily needed to connect to your cluster.

The following snipped can be used to connect:

```
from ray
import os
...
if ray.is_initialized() == False:
   service_host = os.environ[" RAY_HEAD_SERVICE_HOST"]
   service_port = os.environ[" RAY_HEAD_SERVICE_PORT"]
   ray.util.connect(f" {service_host}:{service_port}")

# you should now be connected to the cluster
```

DOMINO

Hands-on - Ray Lab 00 & 01

DOMINO

Thank You

www.dominodatalab.com

nikolay.manchev@dominodatalab.com

www.linkedin.com/in/nikolaymanchev/

@nikolaymanchev