The assumptions behind Machine Learning

... and how they affect your results

Danilo Ferreira de Lima

European XFEL

`danilo.enoque.ferreira.de.lima@xfel.eu`

April 2024

The path ahead

- Focus on the *assumptions* in Machine Learning.
- By choosing assumptions carefully, we can be sure the method works under the conditions we expect.
- These assumptions are closely related to the uncertainties we will estimate.

What is Machine Learning?

What we do *not* want:

(From XKCD)

- Machine Learning requires understanding of the math behind it.
- If you cannot understand it, I would not trust it!
- The mathematical concepts are very simple, but carry *many assumptions*!
- Be careful: data does not care about your assumptions.

A bit of history ...

Decade	Summary
<1950s	Statistical methods are discovered and refined.
1950s	Pioneering machine learning research is conducted using simple algorithms.
1960s	Bayesian methods are introduced for probabilistic inference in machine learning.[1]
1970s	'AI Winter' caused by pessimism about machine learning effectiveness.
1980s	Rediscovery of backpropagation causes a resurgence in machine learning research.
1990s	Work on Machine learning shifts from a knowledge-driven approach to a data-driven approach. Scientists begin creating programs for computers to analyze large amounts of data and draw conclusions – or "learn" – from the results.[2] Support-vector machines (SVMs) and recurrent neural networks (RNNs) become popular.[3] The fields of computational complexity via neural networks and super-Turing computation started.[4]
2000s	Support-Vector Clustering[5] and other kernel methods[6] and unsupervised machine learning methods become widespread.[7]
2010s	Deep learning becomes feasible, which leads to machine learning becoming integral to many widely used software services and applications.

- All methods rely on strong theorems on the underlying statistics of the data.

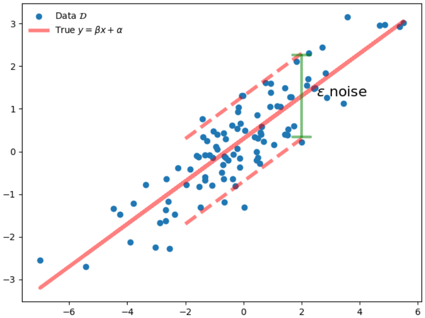
Bayes Theorem: reinterpreting probabilities

$$\underbrace{P(\mathrm{hypothesis}|\mathrm{data})}_{\textit{posterior}} \quad P(\mathrm{data}) = \underbrace{P(\mathrm{data}|\mathrm{hypothesis})}_{\textit{likelihood}} \quad \underbrace{P(\mathrm{hypothesis})}_{\textit{prior}}$$

- So far: probabilities $\rightarrow$ experiment can be repeated *in the same way*.
- What is the probability that the Earth is destroyed?
- The Bayesian view of probability: the degree of *belief* that an event will happen.
  - Make a prior assumption about the event;
  - Calculate the likelihood it will happen based on data;
  - Extract the posterior information with an updated degree of belief.
  - $P(\mathrm{data})$ is the probability this data happens considering all hypotheses. It is often taken as a normalization term.

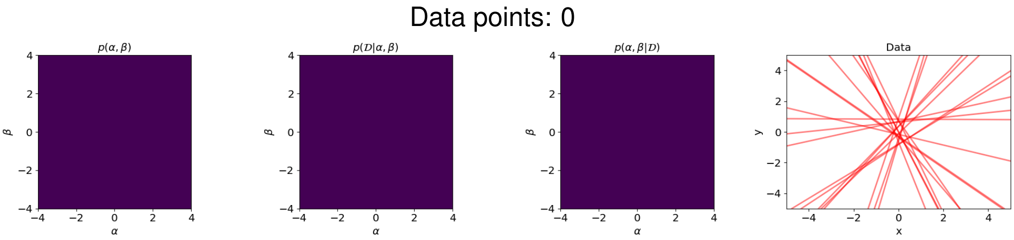
How does this help?!

- Past data $\mathcal{D} \to$ features $\boldsymbol{x}_i$ and y_i.
- Given a new $\boldsymbol{x}'$, what is y'?
- *Assume* $y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) + \epsilon$.
  - ϵ is zero-mean Gaussian noise.
- Example: $y = f(x) = \beta x + \alpha + \epsilon$.
  - In this example, $\theta = (\alpha, \beta)$.



Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

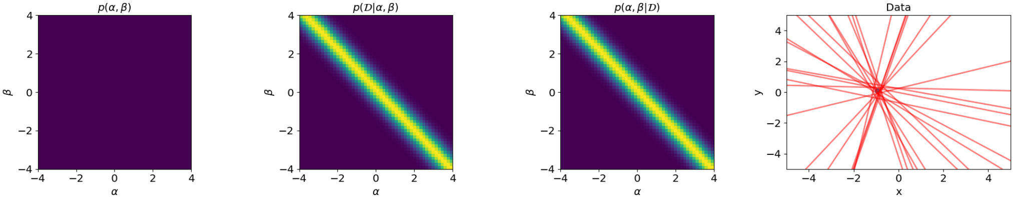
Posterior: updated knowledge.

Samples from the posterior.

Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.

Data points: 1



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

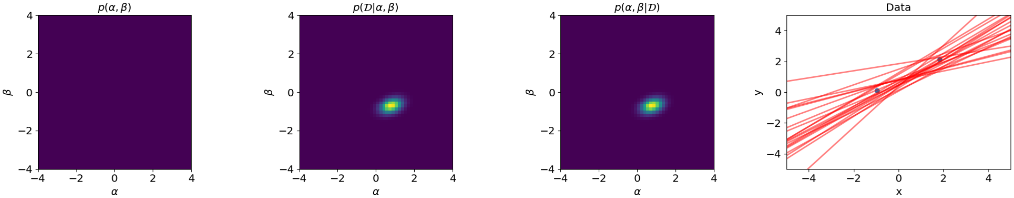
Posterior: updated knowledge.

Samples from the posterior.

Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.

Data points: 2



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

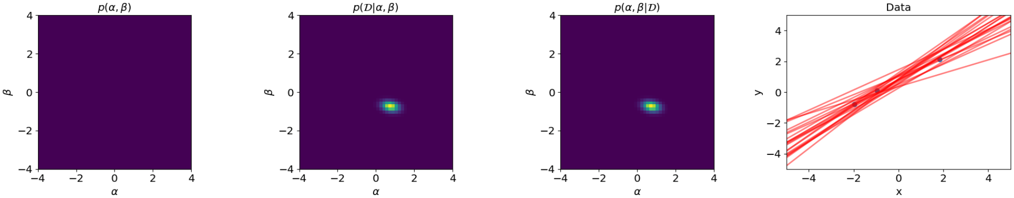
Posterior: updated knowledge.

Samples from the posterior.

Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.

Data points: 3



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

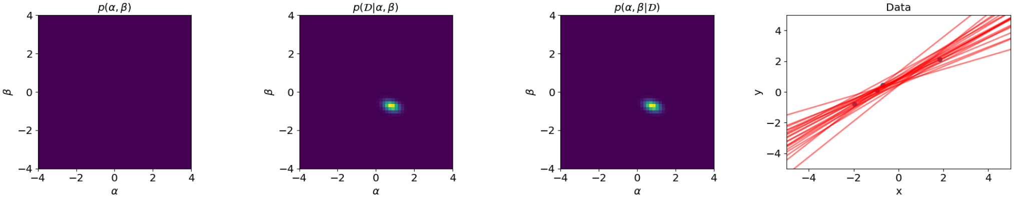
Posterior: updated knowledge.

Samples from the posterior.

Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.

Data points: 4



Prior: what α and β make sense?

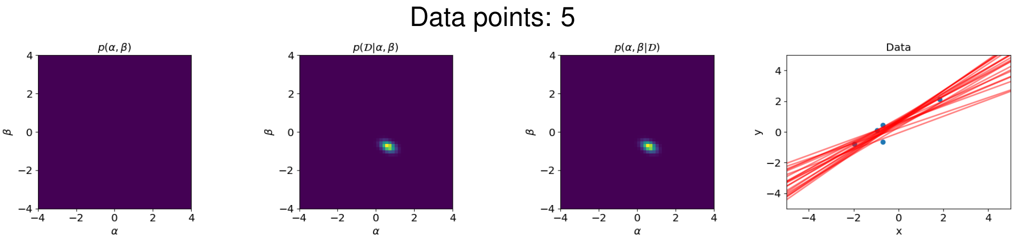
Likelihood: what α and β fit the data?

Posterior: updated knowledge.

Samples from the posterior.

Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

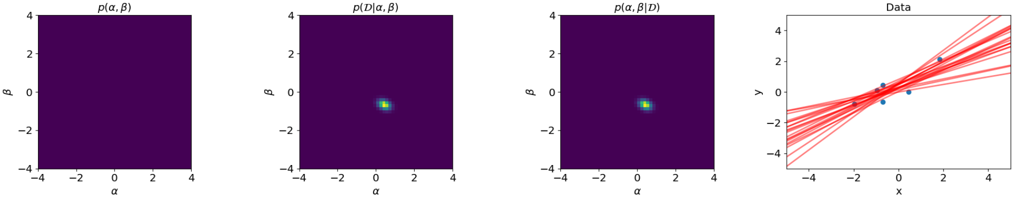
Posterior: updated knowledge.

Samples from the posterior.

Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.

Data points: 6



Prior: what α and β make sense?

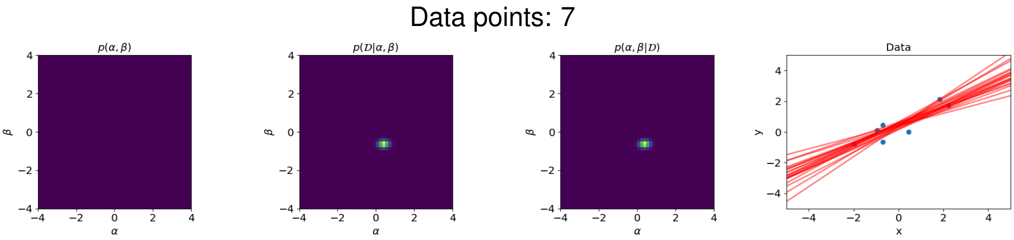
Likelihood: what α and β fit the data?

Posterior: updated knowledge.

Samples from the posterior.

Fitting a line with Bayes Theorem

- As more data is added: update the posterior with more knowledge.



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

Posterior: updated knowledge.

Samples from the posterior.

In general ...

- Best parameters *if the noise is Gaussian*: minimize the sum of squared-error between prediction and target values ($\mathcal{L}$).

$$
\begin{aligned}
p(\boldsymbol{\theta}|\mathcal{D}) &= \frac{p(\mathcal{D}|\boldsymbol{\theta})\,p(\boldsymbol{\theta})}{p(\mathcal{D})} \\
p(\mathcal{D}|\boldsymbol{\theta}) &= p_{\text{Gaussian}}(y|\text{mean} = f_{\boldsymbol{\theta}}(\boldsymbol{x}), \sigma = \sigma_\epsilon) \\
p(\boldsymbol{\theta}) &= \text{constant} \\
\mathcal{L}(\boldsymbol{\theta}) &= -\log p(\boldsymbol{\theta}|\mathcal{D}) \\
\mathcal{L}(\boldsymbol{\theta}) &= \sum_i \frac{1}{2\sigma_\epsilon^2}\left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\right)^2 + \text{cte.}
\end{aligned}
$$

But how do I choose f?

- Cybenko[1] and Hornik[2]: $f_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x}) = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1^T\boldsymbol{x} + b_1) + b_2$ approximates any continuous function for sufficient $\{\boldsymbol{W}, \boldsymbol{b}\}$.
- "Universal approximation theorems": many variations proved.
- *Neural network* $\rightarrow$ stack several of these one after the other.

Universal Approximation Theorem: Fix a continuous function $\sigma : \mathbb{R} \to \mathbb{R}$ (activation function) and positive integers d, D. The function σ is not a polynomial if and only if, for every continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$ (target function), every compact subset K of $\mathbb{R}^d$, and every $\epsilon > 0$ there exists a continuous function $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ (the layer output) with representation

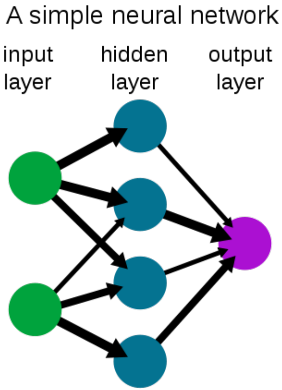
$$f_\epsilon = W_2 \circ \sigma \circ W_1,$$

where W_2, W_1 are composable affine maps and $\circ$ denotes component-wise composition, such that the approximation bound

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \varepsilon$$

holds for any ϵ arbitrarily small (distance from f to f_ϵ can be infinitely small).

(From Wikipedia)

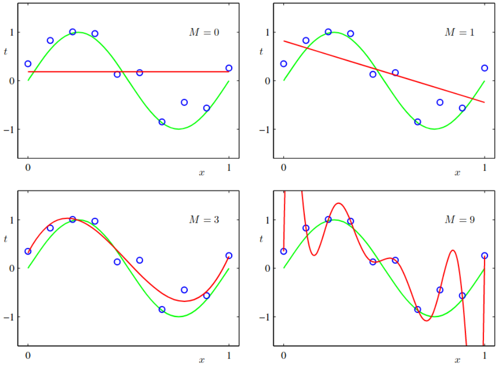


[1] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* 2 (1989), pp. 303–314.
[2] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257.

Are these the only θ?

- Many curves fit the same data $\rightarrow$ control fit complexity.
- Regularization $\rightarrow$ impose prior knowledge on the parameters.
  - E.g.: they should be close to zero.
  - λ controls the regularization strength.

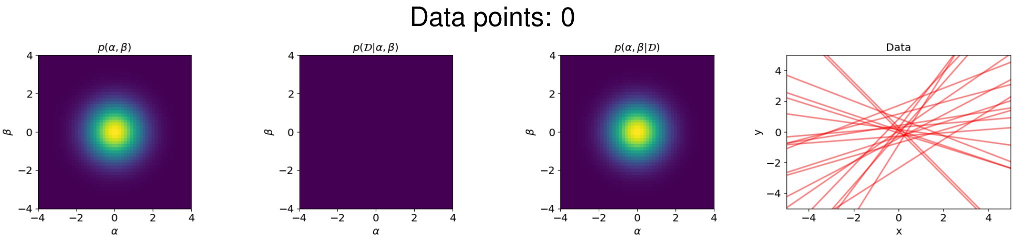
$$\mathcal{L}^{\star}(\theta) = \mathcal{L}(\theta) + \frac{1}{2\lambda^2}\sum_k \theta_k^2$$



(Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006)

Line fit with regularization

- As more data is added: update the posterior with more knowledge.



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

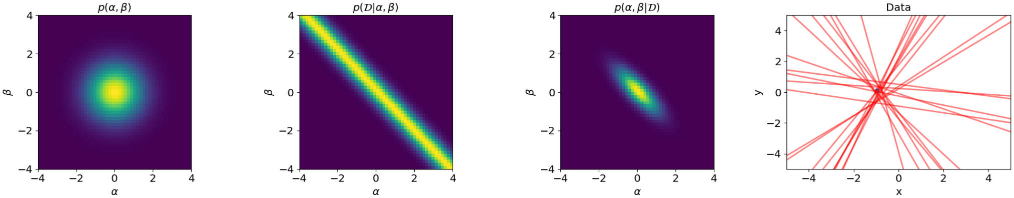
Posterior: updated knowledge.

Samples from the posterior.

Line fit with regularization

- As more data is added: update the posterior with more knowledge.

Data points: 1



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

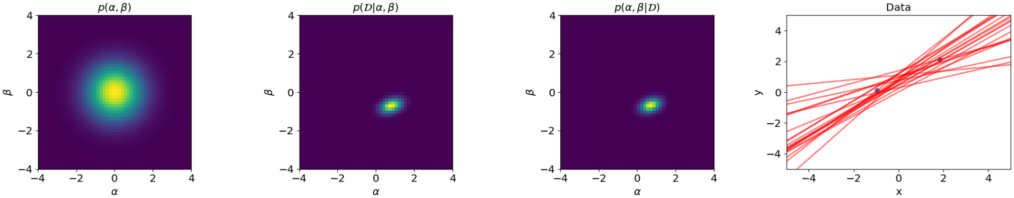
Posterior: updated knowledge.

Samples from the posterior.

Line fit with regularization

- As more data is added: update the posterior with more knowledge.

Data points: 2



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

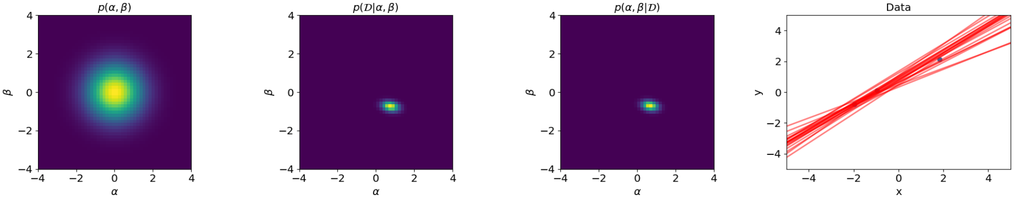
Posterior: updated knowledge.

Samples from the posterior.

Line fit with regularization

- As more data is added: update the posterior with more knowledge.

Data points: 3



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

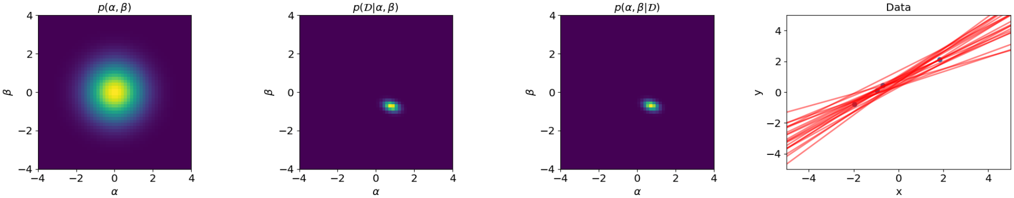
Posterior: updated knowledge.

Samples from the posterior.

Line fit with regularization

- As more data is added: update the posterior with more knowledge.

Data points: 4



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

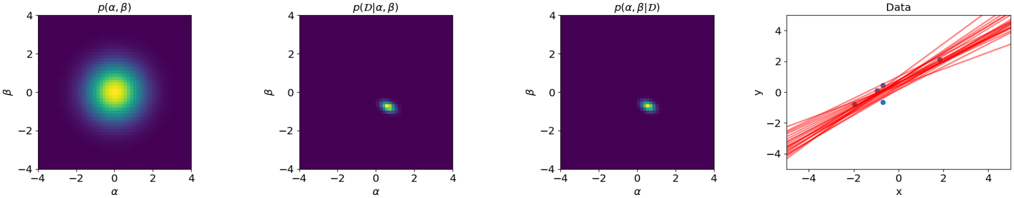
Posterior: updated knowledge.

Samples from the posterior.

Line fit with regularization

- As more data is added: update the posterior with more knowledge.

Data points: 5



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

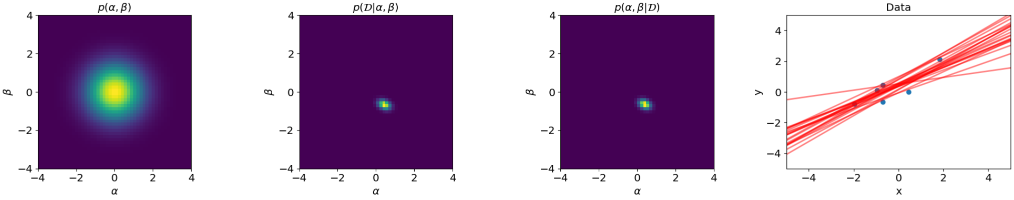
Posterior: updated knowledge.

Samples from the posterior.

Line fit with regularization

- As more data is added: update the posterior with more knowledge.

Data points: 6



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

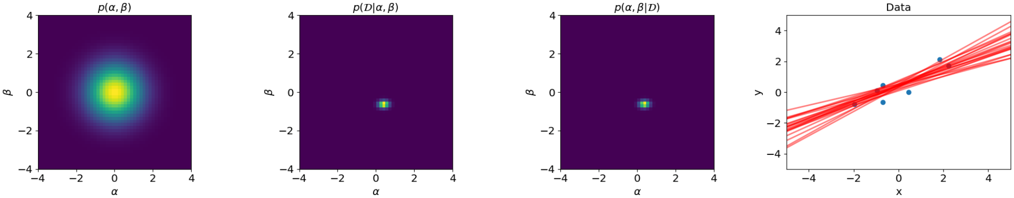
Posterior: updated knowledge.

Samples from the posterior.

Line fit with regularization

- As more data is added: update the posterior with more knowledge.

Data points: 7



Prior: what α and β make sense?

Likelihood: what α and β fit the data?

Posterior: updated knowledge.

Samples from the posterior.

Why does this work?

- Assuming the weights have a Gaussian probability distribution *a priori*.

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\mathcal{D})} \quad \text{Bayes' rule}$$

$$\mathcal{L}^{\star}(\boldsymbol{\theta}) \triangleq -\log p(\boldsymbol{\theta}|\mathcal{D}) \quad \text{Definition}$$

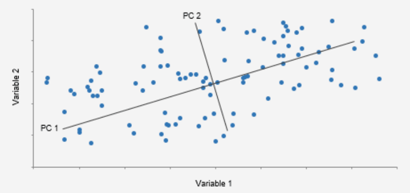
$$\boxed{p(\boldsymbol{\theta}) = p_{\text{Gaussian}}(\theta|0, \lambda)}$$

$$p(\mathcal{D}|\boldsymbol{\theta}) = p_{\text{Gaussian}}(y|\text{mean} = f_{\boldsymbol{\theta}}(\boldsymbol{x}), \sigma = \sigma_{\epsilon})$$

$$\mathcal{L}^{\star}(\boldsymbol{\theta}) = \sum_i \frac{1}{2\sigma_{\epsilon}^2} \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\right)^2 + \frac{1}{2\lambda^2} \sum_k \theta_k^2 + \text{cte.}$$

Representation learning

- Combine variables in a smart way.
- Example: Principal Component Analysis[4].
  - Find eigenvectors of the covariance matrix $C = \mathbb{E}\left[(\boldsymbol{x} - \overline{\boldsymbol{x}})^T (\boldsymbol{x} - \overline{\boldsymbol{x}})\right]$.
  - Rotate in the direction of the eigenvectors to obtain $\boldsymbol{z} = f(\boldsymbol{x})$.
  - Eigenvectors of highest eigenvalues carry more variance.



(From Wikipedia)

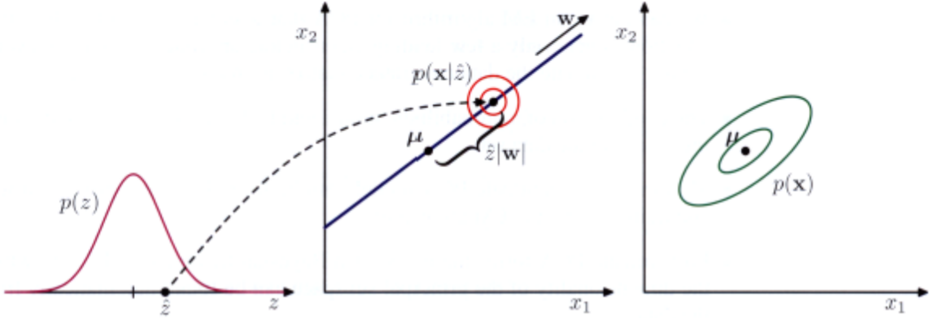
[4] Karl Pearson F.R.S. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

Why does PCA work?

- Assume there is a *latent* space $\boldsymbol{z}$ to which data $\boldsymbol{x}$ can be mapped.
- Assume each variable in $\boldsymbol{z}$ is uncorrelated to each other.

$$
\begin{aligned}
\boldsymbol{x} &= \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \epsilon \\
\boldsymbol{z} &\sim \mathcal{N}(0, \boldsymbol{I}) \\
\epsilon &\sim \mathcal{N}(0, \sigma^2)
\end{aligned}
$$

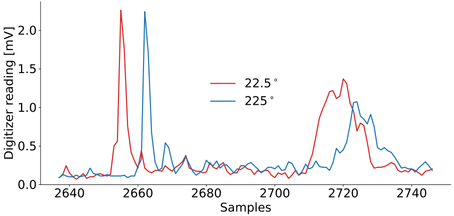
- ϵ is Gaussian noise with std. dev. σ.
- Using Bayes Theorem: $\boldsymbol{W}$ is the matrix of eigenvectors; and $\boldsymbol{\mu} = \mathbb{E}[X]$ (supplementary slides).



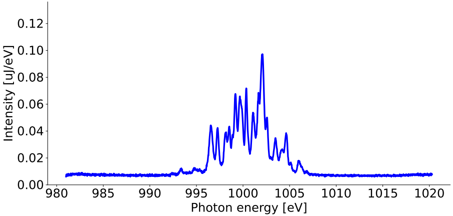
(Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006)

Use-case: Enhancing non-invasive X-ray diagnostics

Photo-Electron Spectrometer (PES)



Grating Spectrometer (GS)

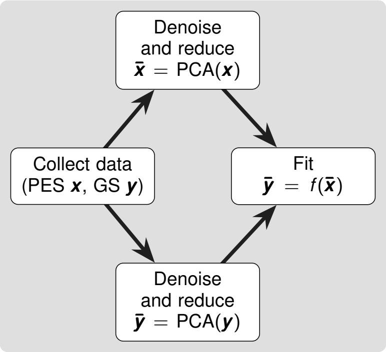


- Low resolution.
- Complex calibration.
- Non-invasive.
- Pulse-resolved.

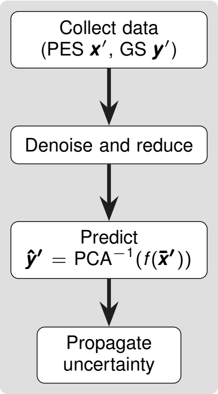
- High resolution.
- Simple calibration
- Invasive.
- Train-resolved.

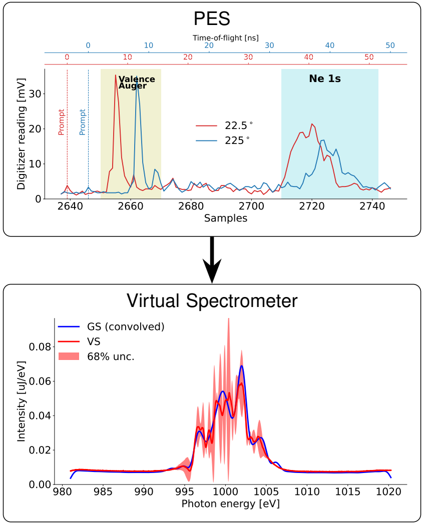
The method

Training



Inference

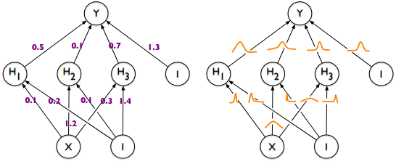




How certain are you about f_θ?

- Data uncertainty $\rightarrow$ different $f_{\boldsymbol{\theta}}$.
- *Assume* $\boldsymbol{\theta}$ are approx. Gaussian.
- Fit mean (μ_θ) and std. deviation (σ_θ) of each θ.
- Proofs and assumptions in[5].
- To optimize $f_{\boldsymbol{\theta}}$: maximize $\mathcal{F}(\boldsymbol{\theta})$.
- Prediction for a new sample $\boldsymbol{x}$:
  - Use many $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu_\theta}, \boldsymbol{\sigma_\theta})$.
  - Prediction $\rightarrow$ mean $f_{\boldsymbol{\theta}}(\boldsymbol{x})$.
  - Uncertainty $\rightarrow$ root-mean-square-error of $f_{\boldsymbol{\theta}}(\boldsymbol{x})$.

Bayesian Neural Networks



$$\mathcal{F}(\boldsymbol{\theta}) = \mathrm{KL}\left[q(\boldsymbol{\theta}|\boldsymbol{\mu_\theta}, \boldsymbol{\sigma_\theta})||p(\boldsymbol{\theta})\right] + \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\mu_\theta}, \boldsymbol{\sigma_\theta})}\left[\log p\,(\mathrm{data}|\boldsymbol{\theta})\right]$$

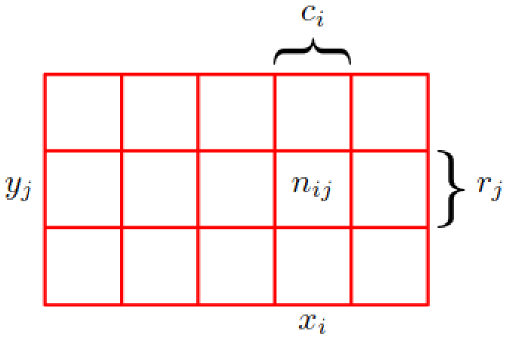
[5] Charles Blundell et al. *Weight Uncertainty in Neural Networks*. 2015. arXiv: `1505.05424 [stat.ML]`.

Summary

- Machine Learning is available to improve your analysis methodology by *sculpting functions* to transform data.
- Take a theory-based approach to Machine Learning!
- It is important to understand the assumptions made in each method and how we can gain an understanding on the data uncertainty.
- Hands-on session includes topics above and, in addition:
  - The kernel method and Support Vector Machines.
  - Gaussian Processes and connection with Neural Networks.
  - Bayesian Optimization.
  - Mixture Models.

Additional material for the hands-on session

Probabilities

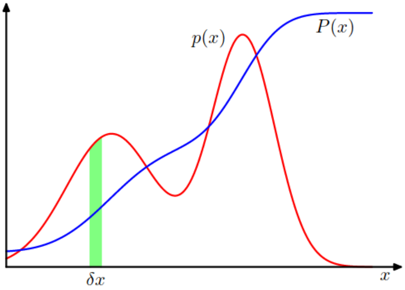


- Do an experiment N times and measure some X and Y.
- We call X and Y *random variables*.
- Their actual values are *samples*: x and y.
- $N \to \infty$: the fraction of times $X = x_i$ and $Y = y_i$ is the probability $P(X = x_i, Y = y_i) \approx \frac{n_{ij}}{N}$.

Probability density

- *Probability density* $\rightarrow$ probability per unit of the random variable.
- Recover probabilities by integrating the probability density.
- Bring over rules from the probabilities:

$$
\begin{aligned}
P(x \in [x_a, x_b]) &= \int_{x_a}^{x_b} p(x)dx \\
p(x) &\geq 0 \\
\int_{-\infty}^{\infty} p(x) &= 1 \\
p(x, y) &= p(x|y)\,p(y) \\
p(x) &= \int_{-\infty}^{\infty} p(x, y)dy
\end{aligned}
$$



Expectations

- Expectation $\rightarrow$ weighted mean with weights given by the probabilities.
- Mean of a random variable X: $\mathbb{E}[X]$.
- Variance of a random variable X: $var[X]$.

$$
\begin{aligned}
\mathbb{E}[X] &= \int_{-\infty}^{\infty} xp(x)dx \\
\mathbb{E}[X] &\approxeq \frac{1}{N}\sum_{n=1}^{N} x_n \\
var[X] &= \mathbb{E}[(X - \mathbb{E}[X])^2]
\end{aligned}
$$

Covariance

- Extend the concept of variance to multiple variables.

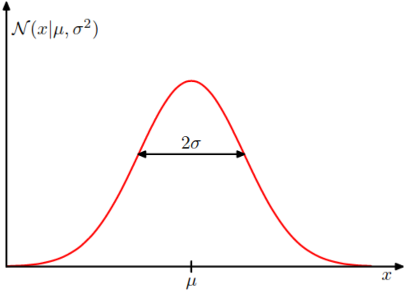
$$cov[\boldsymbol{X}, \boldsymbol{Y}] = \mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])\,(\boldsymbol{Y} - \mathbb{E}[\boldsymbol{Y}])^T]$$

- The covariance normalized by the variance gives us the correlation coefficient.
- Correlation coefficient between -1 and 1 $\rightarrow$ *on average* when X grows, does Y also grow?

$$\rho_{XY} = \frac{cov[X, Y]}{\sqrt{var[X]var[Y]}}$$

Normal distribution

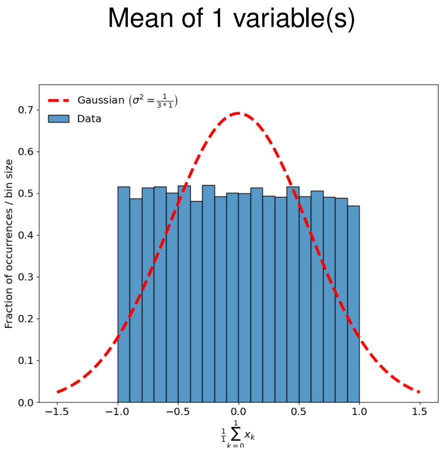
- The normal probability density is often used.
- Values close to the mean parameter, μ, have high probability density.
- Width is related to the standard deviation, σ, parameter.



$$
\begin{aligned}
\mathcal{N}(x|\mu,\sigma^2) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \\
\mathbb{E}[X] &= \mu \\
var[X] &= \sigma^2
\end{aligned}
$$

Central Limit Theorem

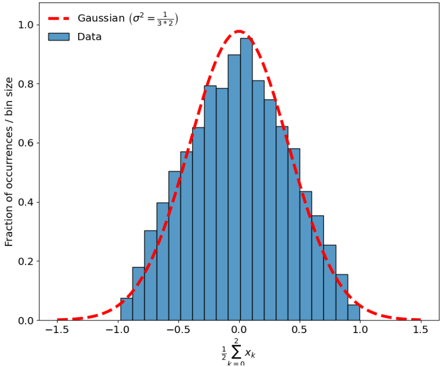
- Experiment with several noise sources.
- The total noise is often almost Gaussian. Why?
- Theorem:
  - *Independent and identically distributed* random variables X_i with the mean 0 and variance σ^2.
  - Mean of N variables $\rightarrow$ Gaussian with variance σ^2/N.
- Right: mean of several uniform distributions with $x \in [-1, 1]$.



Central Limit Theorem

- Experiment with several noise sources.
- The total noise is often almost Gaussian. Why?
- Theorem:
  - *Independent and identically distributed* random variables X_i with the mean 0 and variance σ^2.
  - Mean of N variables $\rightarrow$ Gaussian with variance σ^2/N.
- Right: mean of several uniform distributions with $x \in [-1, 1]$.

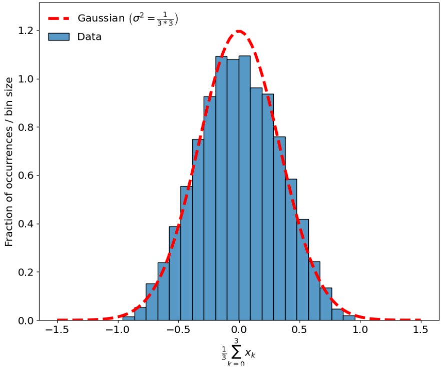
Mean of 2 variable(s)



Central Limit Theorem

- Experiment with several noise sources.
- The total noise is often almost Gaussian. Why?
- Theorem:
  - *Independent and identically distributed* random variables X_i with the mean 0 and variance σ^2.
  - Mean of N variables $\rightarrow$ Gaussian with variance σ^2/N.
- Right: mean of several uniform distributions with $x \in [-1, 1]$.

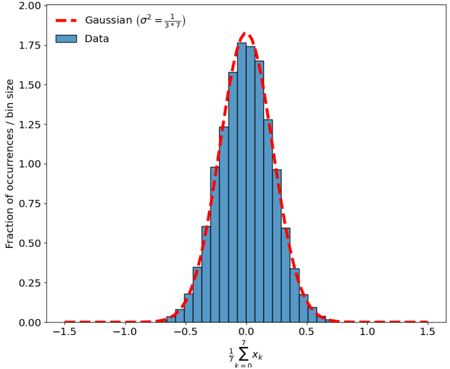
Mean of 3 variable(s)



Central Limit Theorem

- Experiment with several noise sources.
- The total noise is often almost Gaussian. Why?
- Theorem:
  - *Independent and identically distributed* random variables X_i with the mean 0 and variance σ^2.
  - Mean of N variables $\rightarrow$ Gaussian with variance σ^2/N.
- Right: mean of several uniform distributions with $x \in [-1, 1]$.

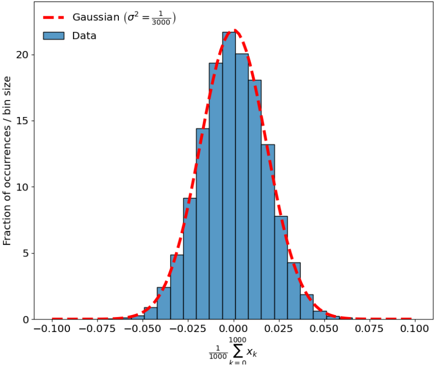
Mean of 7 variable(s)



Central Limit Theorem

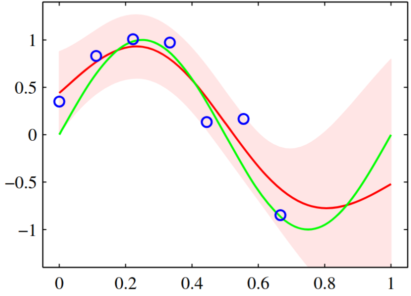
- Experiment with several noise sources.
- The total noise is often almost Gaussian. Why?
- Theorem:
  - *Independent and identically distributed* random variables X_i with the mean 0 and variance σ^2.
  - Mean of N variables $\rightarrow$ Gaussian with variance σ^2/N.
- Right: mean of several uniform distributions with $x \in [-1, 1]$.

Mean of 1000 variables



Gaussian Processes: what if there is no θ?

- *Assume* any pair of $\boldsymbol{y} = f(\boldsymbol{x})$ is Gaussian.
- *Assume a priori* the mean value of $f(\boldsymbol{x})$ is $\mu(\boldsymbol{x})$.
- *Assume a priori* covariance $C(y, y') = C(\boldsymbol{x}, \boldsymbol{x}')$.
- Can calculate the full distribution of $\boldsymbol{y}$ without any parameters!
- Infinitely complex NNs are Gaussian Processes[6].
- Disadvantage: high computational complexity.



(Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006)

[6] Radford M. Neal. "Priors for Infinite Networks". In: *Bayesian Learning for Neural Networks*. New York, NY: Springer New York, 1996, pp. 29–53.

How can it be done?

Simplification: $\mu(\boldsymbol{x}) = 0$.
f is Gaussian between the training data $\boldsymbol{x}$ and a new probe point $\boldsymbol{x}^\star$.
von Mises [1964]: we can estimate distribution of $f(\boldsymbol{x}^\star)$ given $\boldsymbol{x}$.

$$
\begin{aligned}
f(\boldsymbol{x}) &\sim \mathcal{GP}\left(0, C(\boldsymbol{x}, \boldsymbol{x}')\right) \\
\begin{bmatrix} f(\boldsymbol{x}) \\ f(\boldsymbol{x}^\star) \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} C_{\boldsymbol{xx}} & C_{\boldsymbol{xx}^\star} \\ C_{\boldsymbol{x}^\star\boldsymbol{x}} & C_{\boldsymbol{x}^\star\boldsymbol{x}^\star} \end{bmatrix}\right)
\end{aligned}
$$

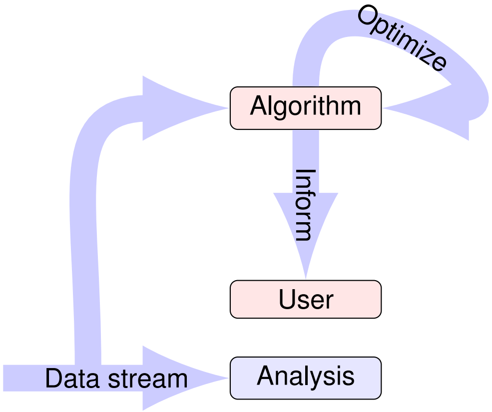
Theorem (von Mises [1964]): if $\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix}^{-1}\right)$ is a jointly Gaussian variable, then the conditional distribution of x given y is

$$
\begin{aligned}
x|y &\sim \mathcal{N}\left(\mu_x - A^{-1}C(y - \mu_y), A^{-1}\right), \text{ or ...} \\
x|y &\sim \mathcal{N}\left(\mu_x + CB^{-1}(y - \mu_y), A - CB^{-1}C^T\right)
\end{aligned}
$$

Estimate mean and uncertainty at a probe $\boldsymbol{x}^\star$:

$$
f(\boldsymbol{x}^\star)|f(\boldsymbol{x}) \sim \mathcal{N}\left(C_{\boldsymbol{x}^\star\boldsymbol{x}} C_{\boldsymbol{xx}}^{-1} f(\boldsymbol{x}), C_{\boldsymbol{x}^\star\boldsymbol{x}^\star} - C_{\boldsymbol{x}^\star\boldsymbol{x}} C_{\boldsymbol{xx}}^{-1} C_{\boldsymbol{xx}^\star}\right).
$$

An example: automatizing SFX



- SFX analysis pipeline has several parameters.
- Online feedback improves the efficiency of the beamtimes.
- Attempt to find parameters that *maximize fraction of indexed frames*.
- Improved parameters found? $\rightarrow$ update the standard pipeline.
- Bayesian Optimization.

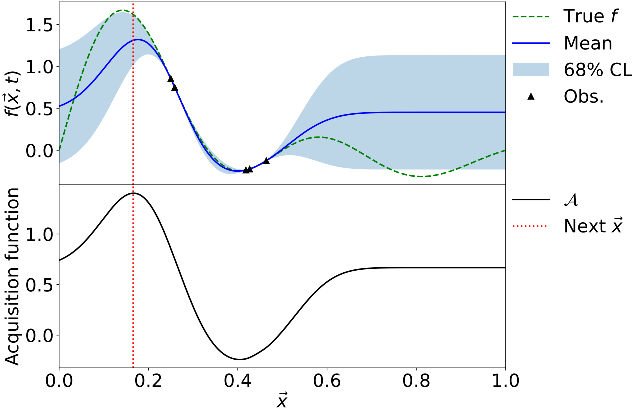
Bayesian Optimization: Initialize

- Run analysis ($\times n$) and store:

Parameter	Objective
$\vec{x}_1$	$f(\vec{x}_1)$
$\vdots$	$\vdots$
$\vec{x}_n$	$f(\vec{x}_n)$

Bayesian Optimization: Fit

- Fit $\hat{f}(\vec{x})$ and uncertainty.
- $\vec{x}_{n+1} = \arg\max_{\vec{x}} \mathcal{A}(\vec{x})$.



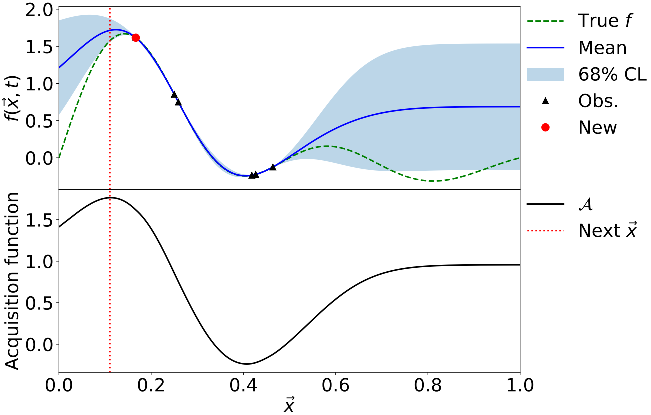
Bayesian Optimization: Probe

- Run analysis at $\vec{x}_{n+1}$ and store:

Parameter	Objective
$\vec{x}_1$	$f(\vec{x}_1, \mathcal{D})$
$\vdots$	$\vdots$
$\vec{x}_{n+1}$	$f(\vec{x}_{n+1}, \mathcal{D})$

Bayesian Optimization: Re-fit

- Fit $\hat{f}(\vec{x})$ and uncertainty.
- $\vec{x}_{n+2} = \arg\max_{\vec{x}} \mathcal{A}(\vec{x})$.



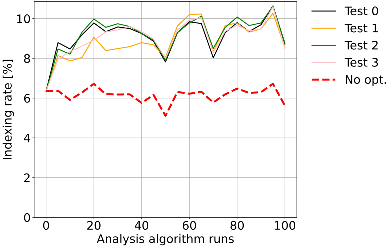
Bayesian Optimization: Probe

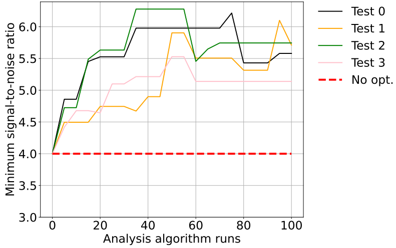
- Run analysis at $\vec{x}_{n+2}$ and store:

Parameter	**Objective**
$\vec{x}_1$	$f(\vec{x}_1, \mathcal{D})$
$\vdots$	$\vdots$
$\vec{x}_{n+1}$	$f(\vec{x}_{n+1}, \mathcal{D})$
$\vec{x}_{n+2}$	$f(\vec{x}_{n+2}, \mathcal{D})$

Performance

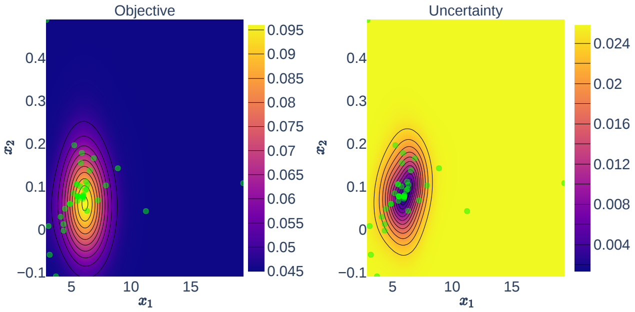
- Very stable and requires few iterations to converge.
- For a quick example: simulated data flowing in AGIPD tuning det. centre, min. SNR and detector-sample distance.





Building trust

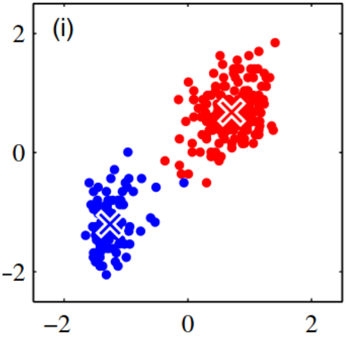
- More is not better: only the *uncertainties* can tell!
- Several methods in use do not provide them → can we know when they fail?
- Tell the users our limitations!



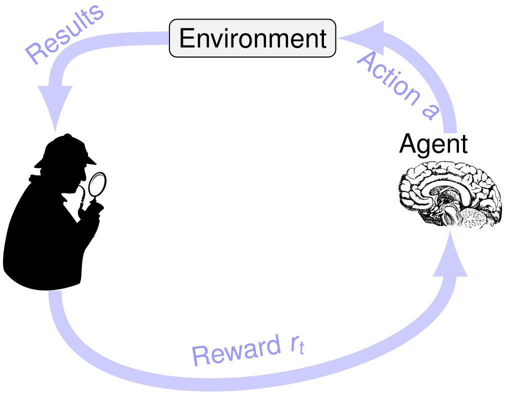
Mixture models

- Data produced from K different *templates*.
- Templates/clusters belong to a discrete set.
- Gaussian Mixture Model samples formalized as:
  - draw an integer number k identifying the cluster;
  - sample from Gaussian with parameters $p_k = (\mu_k, \sigma_k)$.
- Methods to find clusters:
  - Heuristic.
  - Expectation Maximization (EM).
  - Approximate variational inference (use prior knowledge and automatically choose the number of clusters).

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z}) p(\boldsymbol{x}|\boldsymbol{z}) = \sum_k \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k)$$



Reinforcement Learning



- Environment $\rightarrow$ analysis pipeline.
- Model-free RL $\rightarrow$ environment-independent.
  - Input = last + noise (*perform action*).
  - Was there an improvement (*collect reward*)?
  - Update agent.
- Objective: maximize total returns G after T attempts.
  - At $t = T$, reset state to the best.

$$G = \sum_{t=1}^{T} \gamma^t r_t$$

On-policy Reinforcement Learning

- How to optimize the actor?
  - Maximize returns $\rightarrow$ move parameters in the direction of $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\pi(\tau|\boldsymbol{\theta})}\left[G(\tau)\right]$.
- Note: it is hard to calculate $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\pi(\tau|\boldsymbol{\theta})}\left[\cdot\right] \rightarrow$ cannot move $\nabla_{\boldsymbol{\theta}}$ inside $\mathbb{E}_{f(\boldsymbol{\theta})}$, because of $\boldsymbol{\theta}$!
- Williams[7] showed that:
  - $\mathbb{E}_{\pi(\tau|\boldsymbol{\theta})}\left[\nabla_{\boldsymbol{\theta}} \log \pi(\tau|\boldsymbol{\theta}) G(\tau)\right]$ is an unbiased estimator for $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\pi(\tau|\boldsymbol{\theta})}\left[G(\tau)\right]$.
  - Probability of taking a set of actions $\tau = \{(s_1, a_1), \ldots, (s_n, a_n)\}$ is $\pi(\tau)$.
  - Parameters of the neural network $\boldsymbol{\theta}$.
- Many algorithm variations of the objective to improve the returns estimate.

[7] R. J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8 (1992), pp. 229–256.

Example RL algorithm: REINFORCE

```
procedure MAIN
    while True do
        memory ← Explore and collect rewards()
        Learn(memory)

procedure LEARN
    G ← calculate returns(rewards)
    ℒ ← Mean [G log π(states)]
    Minimize ℒ

procedure EXPLORE AND COLLECT REWARDS
    memory ← {}
    for i in 0...N episodes do
        state ← initial state
        for t in 0...T steps do
            action ← RandomSample(π_θ(a|s))
            state ← state + action
            reward ← GetReward(state)
        memory ← memory +
         rewards, states, actions
    return memory
```

Off-policy RL

- On-policy RL is data-inefficient.
- Bellman[8] established an optimality condition.
    - Self-consistency of the expected returns.
- Off-policy methods learn to use data from old policies as well.
- Need stabilization methods to avoid diverging from the optimality condition.

Definition

$$Q^\pi(s_t, a_t) \triangleq \mathbb{E}_{\pi, i \geq t}\left[G_t | s_t, a_t\right]$$

Bellman equation

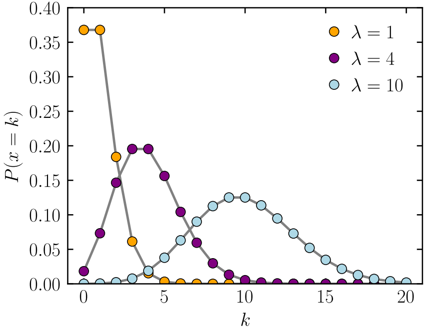
$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}\left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1}}\left[Q^\pi(s_{t+1}, a_{t+1})\right]\right]$$

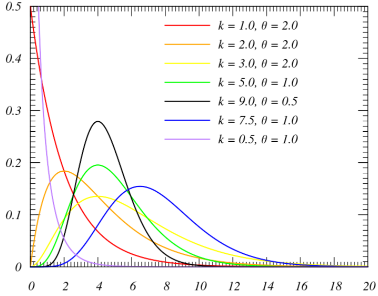
[8] Richard Bellman. “On the Theory of Dynamic Programming”. In: *Proceedings of the National Academy of Sciences* 38.8 (1952), pp. 716–719.

Supplementary material

Estimating probability distributions

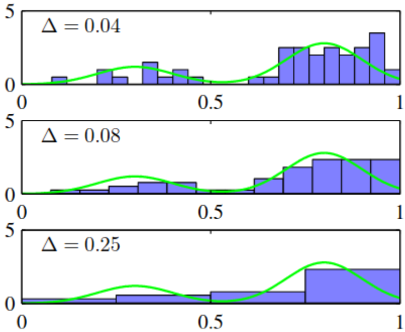
- How can we know if the data follows a given distribution?
- We can try to fit the data with a parametrised pdf function.
- Several pdf parametrizations with different properties.





Histograms

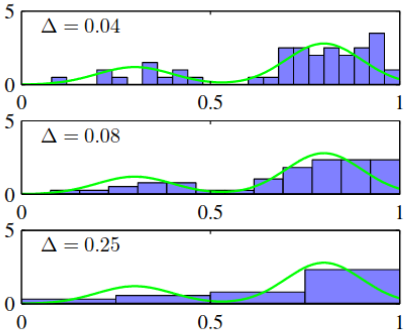
- A non-parametric solution is to just count how many times a variable appears in several ranges.
- Issue: very dependent on the choice of ranges (bins).
- May provide a skewed view of the pdf when counts are low.
- Becomes very memory-consuming for a pdf of multiple variables.



Kernel density estimation

- An alternative: for each observed data point x_n, use a smoothing function (the kernel) $k(\cdot)$ and sum up the effect of each data point.
- The smoothing reduces the discontinuities from histograms.
- The choice of the kernel width h can be made following some rules of thumb (see Scott's method, for an example) based on the dimension and amount of data.

$$p(x) \approxeq \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}K\left(\frac{x-x_n}{h}\right)$$



And where do we get those derivatives?

$$f = A\sigma(W^T x + b)$$

$$g_l(y) = \sigma(y) \qquad g_r(y) = Ay$$

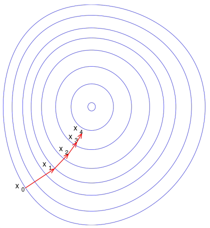
$$h_l(y) = y^T x \qquad h_r(y) = y + b$$

- Missing piece: derivative of $f(\theta)$.
- $\frac{f(\theta+\epsilon)-f(\theta)}{\epsilon}$ for small $\epsilon \rightarrow$ large numerical errors.
- Store table of derivatives and use the chain rule.
- Automatic differentiation ($\neq$ symbolic differentiation!).
- Back-propagation:
  - *Forward*-propagate inputs $\rightarrow$ network result.
  - *Backwards*-propagate outputs in each step for ∇f.
  - *All you need is the chain rule.*

$$\frac{\partial f(W)}{\partial W} = \frac{\partial g_r(y)}{\partial y}\Big|_{y=g_l} \times \frac{\partial g_l(y)}{\partial y}\Big|_{y=h_r} \times \frac{\partial h_r(y)}{\partial y}\Big|_{y=h_l} \times \frac{\partial h_l(y)}{\partial y}\Big|_{y=h_r}$$

How to find θ?

- We need to find $\boldsymbol{\theta}$ which minimizes $\mathcal{L}$.
- Assume $\boldsymbol{\theta} = \boldsymbol{\theta}_0$; choose $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}$ that reduces $\mathcal{L}$; repeat.
- Many papers on how to best approximate $\boldsymbol{H} \rightarrow$ BFGS[9], Adam[10], ...



$$\mathcal{L}(\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}) \approx \mathcal{L}(\boldsymbol{\theta}_0) + \Delta\boldsymbol{\theta}^T \, \nabla_{\boldsymbol{\theta}} \mathcal{L}|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} + \frac{1}{2} \Delta\boldsymbol{\theta}^T \, \boldsymbol{H}(\boldsymbol{\theta}_0) \, \Delta\boldsymbol{\theta} \quad \leftarrow \textit{Taylor series}$$

$$\text{Gradient} \rightarrow \nabla_{\Delta\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} + \boldsymbol{H}(\boldsymbol{\theta}_0) \, \Delta\boldsymbol{\theta}$$

$$\text{In optimum} \rightarrow \nabla_{\Delta\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}) = 0$$

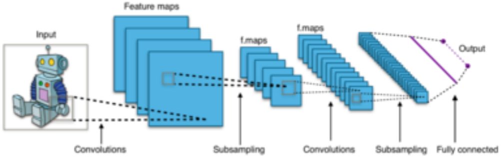
$$\text{Step towards} \rightarrow \boxed{\Delta\boldsymbol{\theta} = -\boldsymbol{H}(\boldsymbol{\theta}_0)^{-1} \, \nabla_{\boldsymbol{\theta}} \mathcal{L}|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}}$$

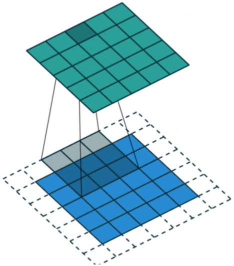
$$\text{When } \boldsymbol{H} \approx \eta^{-1} \boldsymbol{I} \rightarrow \Delta\boldsymbol{\theta} = -\eta \, \nabla_{\boldsymbol{\theta}} \mathcal{L}|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}$$

[9] C. G. BROYDEN. "The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations". In: *IMA Journal of Applied Mathematics* 6.1 (Mar. 1970), pp. 76–90.
[10] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

But images are too big!

- Images with many pixels require a lot of parameters $\boldsymbol{\theta}$. How can we reduce them?
- Assume important features of the images are mostly locally correlated.
- Convolutional Neural Networks substitute $\boldsymbol{W}^T\boldsymbol{x}$ with $\boldsymbol{W} \star \boldsymbol{x}$.
- $\star \rightarrow$ convolution $\rightarrow$ local filter.





What if we want to classify data?

- Data $\mathcal{D} \rightarrow \boldsymbol{x}_i$ and labels $y_i \in \{1, \ldots, N\}$.
- $f_{\boldsymbol{\theta},k}$ shall output the probability $p(C_k|\boldsymbol{x}_i)$ that $\boldsymbol{x}_i$ belongs to each class C_k.
- *Assume* all $\boldsymbol{\theta}$ are equally likely.
- Use *Bayes' rule*:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

- Choose $\boldsymbol{\theta}$ that minimizes $\mathcal{L}(\boldsymbol{\theta})$.

$$f_{\boldsymbol{\theta},k}(\boldsymbol{x}_i) = p(C_k|\boldsymbol{x}_i)$$

$$t_{ik} = \begin{cases} 1, \text{if } y_i = k \\ 0, \text{otherwise} \end{cases}$$

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i,k} p(C_k|\boldsymbol{x}_i)^{t_{ik}}$$

$$p(\boldsymbol{\theta}) = \text{cte.}$$

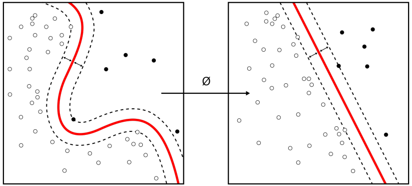
$$\mathcal{L}(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}|\mathcal{D})$$

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i,k} t_{ik} \log\left(f_{\boldsymbol{\theta},k}(\boldsymbol{x}_i)\right) + \text{cte.}$$

$$-\sum_{k} p(A_k) \log p(B_k) \rightarrow \text{cross-entropy}(A, B)$$

Kernel methods

- Neural networks: expand non-linearities of functions.
- Different approach: choose $\Phi : \boldsymbol{x} \rightarrow \Phi(\boldsymbol{x})$, such that $\boldsymbol{y} = \sum_i \theta_i \Phi(\boldsymbol{x})$.
- Can be reformulated with a *kernel* function $k(\boldsymbol{x}, \boldsymbol{x}') = \Phi(\boldsymbol{x})^T \Phi(\boldsymbol{x}')$ (dual formulation in[a]).
- Support Vector Machines (SVMs): transform a non-linear problem into a linear one.



(Wikipedia)

[a]Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

PCA in maths (I)

Assumptions:

$$\begin{aligned} p(\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{z}|0, I) \\ p(\boldsymbol{x}|\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2\boldsymbol{I}) \\ p(\boldsymbol{x}) &= \int p(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})\, d\boldsymbol{z} \end{aligned}$$

Since every term is Gaussian, $p(\boldsymbol{x})$ is also Gaussian:

$$\begin{aligned} p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{C}, \sigma) &= \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{C}) \\ \boldsymbol{\mu} &= \mathbb{E}[\boldsymbol{X}] \\ \boldsymbol{C} &= \boldsymbol{W}\boldsymbol{W}^T + \sigma^2\boldsymbol{I} = cov[\boldsymbol{X}] \end{aligned}$$

PCA in maths (II)

We can find the optimum $\boldsymbol{W}$ and $\boldsymbol{\mu}$ by maximizing the log-likelihood over all N data points:

$$
\begin{aligned}
\log p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{W}, \sigma) &= \sum_i^N p(\boldsymbol{x}_i|\boldsymbol{W}, \boldsymbol{\mu}, \sigma) \\
&= -\frac{N}{2}|\boldsymbol{C}| - \frac{1}{2}\sum_i^N (\boldsymbol{x}_i - \boldsymbol{\mu})\boldsymbol{C}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})
\end{aligned}
$$

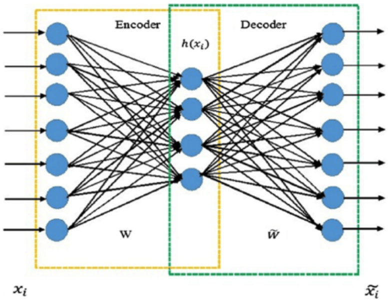
Setting $\nabla \log p = 0$ (Tipping and Bishop [1999]):

$$
\boldsymbol{W} = \boldsymbol{U}(\boldsymbol{L} - \sigma^2 \boldsymbol{I})^{1/2}\boldsymbol{R}
$$

- $\boldsymbol{U}$ is a matrix with a subset of eigenvectors in the columns;
- $\boldsymbol{L}$ is a diagonal matrix of eigenvalues;
- $\boldsymbol{R}$ is an arbitrary orthogonal matrix;
- the likelihood is highest if the eigenvectors have the highest eigenvalues.

Auto-encoders: PCA on steroids

- PCA is linear. How to model non-linear transformations?
- Kernel PCA: use the kernel trick.
- Another way: auto-encoders.
  - Map the input data to itself.
  - Force data compression: reduce dimension in intermediate layer.
- Linear NN $\equiv$ PCA[11].
- Non-linear NN $\rightarrow$ non-linear representation.



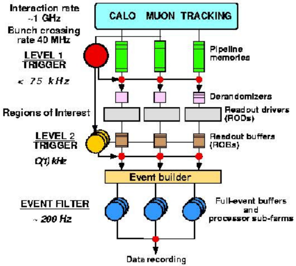
- *Assuming* Gaussian errors, minimize:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x} \in \text{data}} \left[f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{x} \right]^2$$

[11] H. Bourlard and Y. Kamp. “Auto-association by multilayer perceptrons and singular value decomposition”. In: *Biol. Cybern.* 59 (1988), pp. 291–294.

Example: The ATLAS Trigger system at the LHC

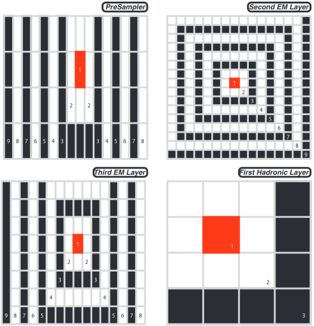
- ATLAS detector at the LHC: protons are collided and only a fraction of collisions are relevant for physics research.
- Task: select collision events from O(MHz) events to save data at a rate of only O(1 Hz).
- Reliable simulation available → allows for supervised learning.



ATLAS trigger system overview [outdated now]

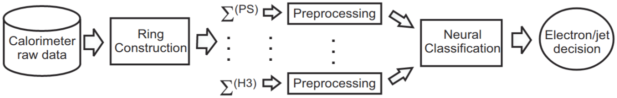
Data pre-processing

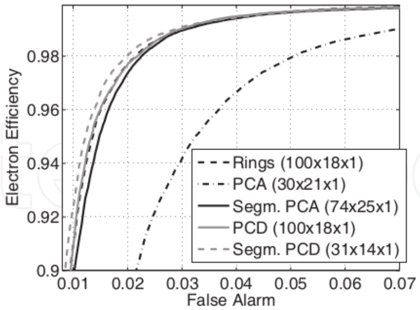
- Fast decisions: use (approx.) symmetry in the input image!
- Rings of energy in each detector layer as the input $\rightarrow$ avoid large inputs, complex neural networks and aim for fast processing.



Electron Neural Network preselection

- PCA to select most relevant components[12] → simpler neural networks with similar performance.





[12] D. de Lima et al. “Signal Processing”. In: ed. by Sebastian Miron. INTECH, 2010. Chap. Segmented Online Neural Filtering System Based On Independent Components Of Pre-Processed Information.