Lectures on Machine Learning

Lecture 1: from artificial intelligence to parameter learning

Stefano Carrazza

TAE2023, 11-12 September 2023

University of Milan and INFN Milan (UNIMI)

N3PDF
Machine Learning • PDFs • QCD

INFN
Istituto Nazionale di Fisica Nucleare

Why lectures on machine learning?

Why lectures on machine learning?

because

- it is an essential set of algorithms for building models in science,

Why lectures on machine learning?

because

- it is an essential set of algorithms for building models in science,
- fast development of new tools and algorithms in the past years,

Why lectures on machine learning?

because

- it is an essential set of algorithms for building models in science,
- fast development of new tools and algorithms in the past years,
- nowadays it is a requirement in experimental and theoretical physics,

Why lectures on machine learning?

because

- it is an essential set of algorithms for building models in science,
- fast development of new tools and algorithms in the past years,
- nowadays it is a requirement in experimental and theoretical physics,
- large interest from the HEP community: *IML*, *conferences*, *grants*.

What expect from these lectures?

What expect from these lectures?

- Learn the basis of machine learning techniques.
- Learn when and how to apply machine learning algorithms.

The talk is divided in two lectures:

Lecture 1 (today)

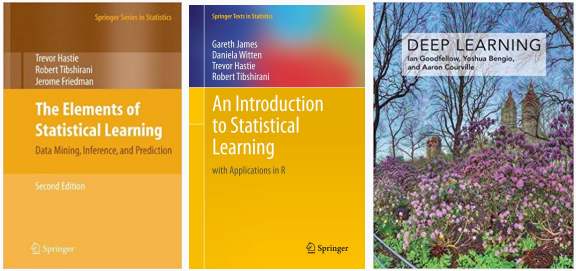
- Artificial intelligence
- Machine learning
- Model representation
- Metrics
- Parameter learning

Lecture 2 (tomorrow)

- Non-linear models
- Beyond neural networks
- Clustering
- Cross-validation
- Hyperparameter tune

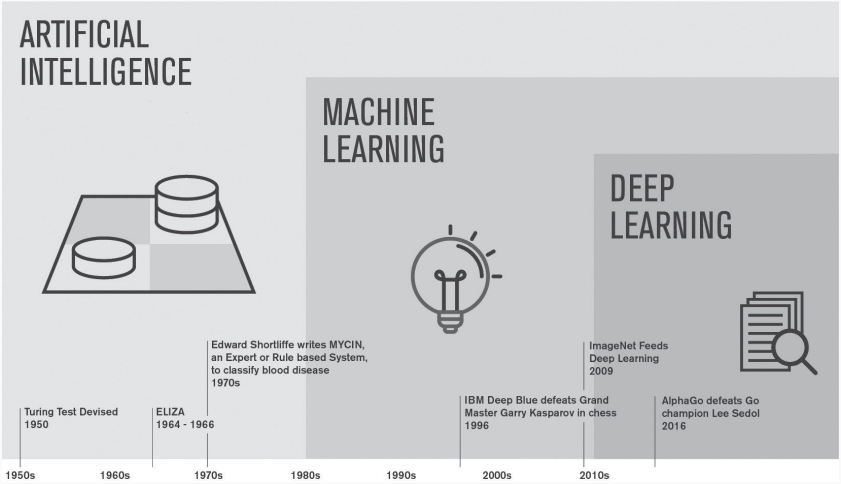
Some references

- *The elements of statistical learning*, T. Hastie, R. Tibshirani, J. Friedman.
- *An introduction to statistical learning*, G. James, D. Witten, T. Hastie, R. Tibshirani.
- *Deep learning*, I. Goodfellow, Y. Bengio, A. Courville.



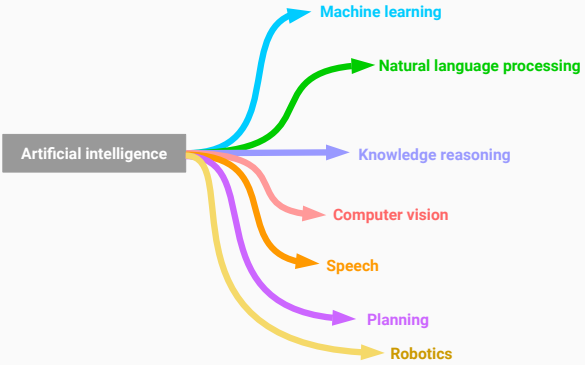
Artificial Intelligence

Artificial intelligence timeline



Defining A.I.

Artificial intelligence (A.I.) is *the science and engineering of making intelligent machines.* (John McCarthy '56)

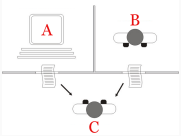


A.I. consist in the development of computer systems to perform tasks commonly associated with intelligence, such as *learning*.

A.I. and humans

There are two categories of A.I. tasks:

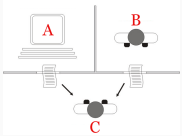
- **abstract and formal:** easy for computers but difficult for humans, *e.g.* play chess (IBM's Deep Blue 1997).
  → *Knowledge-based* approach to artificial intelligence.



A.I. and humans

There are two categories of A.I. tasks:

- **abstract and formal:** easy for computers but difficult for humans, *e.g.* play chess (IBM's Deep Blue 1997).
  → *Knowledge-based* approach to artificial intelligence.



- **intuitive for humans but hard to describe formally:**
  e.g. recognizing faces in images or spoken words.
  → *Concept* capture and generalization

A.I. technologies

Historically, the *knowledge-based* approach has not led to a major success with intuitive tasks for humans, because:

- requires human *supervision* and hard-coded *logical inference rules*.
- lacks of *representation learning* ability.

A.I. technologies

Historically, the *knowledge-based* approach has not led to a major success with intuitive tasks for humans, because:

- requires human *supervision* and hard-coded *logical inference rules*.
- lacks of *representation learning* ability.

Solution:

The A.I. system needs to *acquire its own knowledge*.
This capability is known as **machine learning** (ML).
$\rightarrow$ *e.g.* write a program which learns the task.

Machine learning definition

Machine Learning definition (from T. Mitchell in 1998):

A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance on T, as measured by P, improves with experience E.

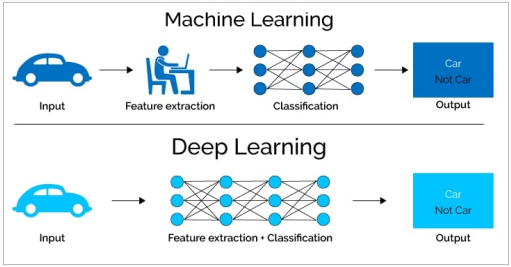
Machine learning definition

Machine Learning definition (from T. Mitchell in 1998):

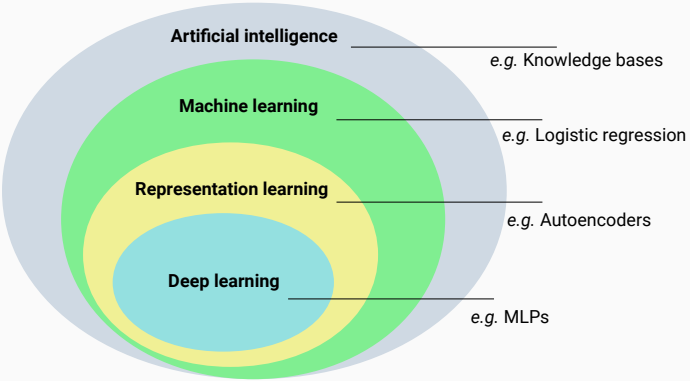
A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance on T, as measured by P, improves with experience E.

Deep Learning

When a representation learning is difficult, ML provides **deep learning** techniques which allow the computer to build complex concepts out of simpler concepts, *e.g.* artificial neural networks.



Venn diagram for A.I.



Machine learning examples

Thanks to work in A.I. and new capability for computers:

- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records

Machine learning examples

Thanks to work in A.I. and new capability for computers:

- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records
- **Intuitive tasks for humans:**
  - Autonomous driving
  - Natural language processing
  - Robotics (reinforcement learning)
  - Game playing (DQN algorithms)

Machine learning examples

Thanks to work in A.I. and new capability for computers:

- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records
- **Intuitive tasks for humans:**
  - Autonomous driving
  - Natural language processing
  - Robotics (reinforcement learning)
  - Game playing (DQN algorithms)

Machine learning examples

Thanks to work in A.I. and new capability for computers:

- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records
- **Intuitive tasks for humans:**
  - Autonomous driving
  - Natural language processing
  - Robotics (reinforcement learning)
  - Game playing (DQN algorithms)
- **Human learning:**
  - Concept/human recognition
  - Computer vision
  - Product recommendation

ML applications in HEP

ML in experimental HEP

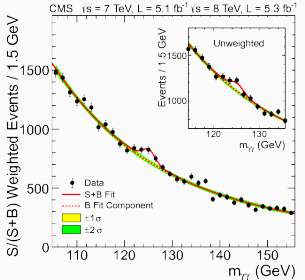
There are many applications in experimental HEP involving the **LHC measurements**, including the **Higgs discovery**, such as:

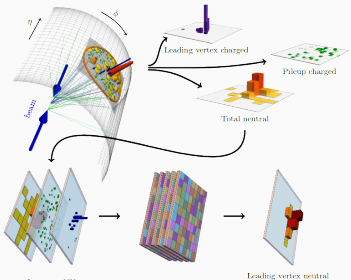
- Tracking
- Fast Simulation
- Particle identification
- Event filtering

ML in experimental HEP

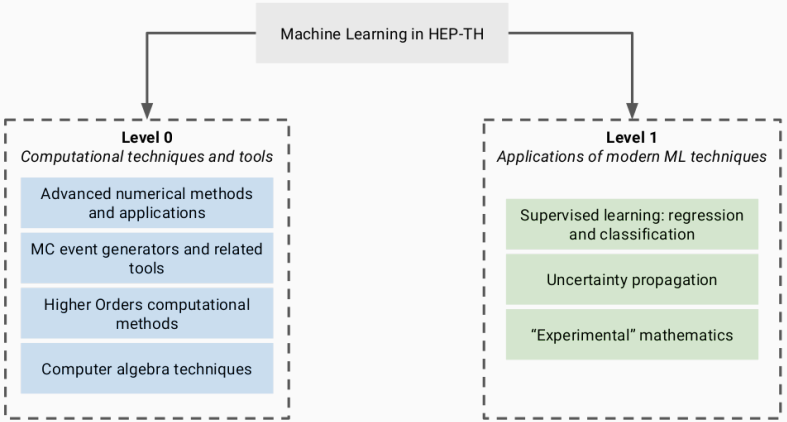
Some remarkable examples are:

- **Signal-background detection:**
  Decision trees, artificial neural networks, support vector machines.
- **Jet discrimination:**
  Deep learning imaging techniques via convolutional neural networks.
- **HEP detector simulation:**
  Generative adversarial networks, *e.g.* LAGAN and CaloGAN.



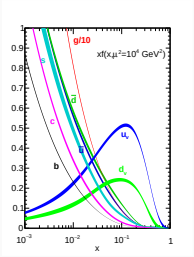


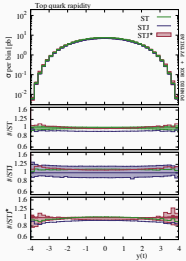
ML in theoretical HEP



ML in theoretical HEP

- **Supervised learning:**
  - The structure of the proton at the LHC
    - parton distribution functions
  - Theoretical prediction and combination
  - Monte Carlo reweighting techniques
    - neural network Sudakov
  - BSM searches and exclusion limits
- **Unsupervised learning:**
  - Jet physics
    - GANs and CycleGANs for jet reconstruction
  - Clustering and compression
    - PDF4LHC15 recommendation
  - Density estimation and anomaly detection
    - Monte Carlo sampling
- **Reinforcement learning:**
  - Jet grooming



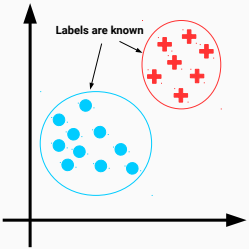


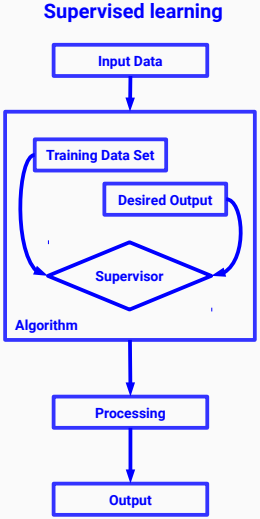
Learning paradigm

Machine learning algorithms

Machine learning algorithms:

- Supervised learning:
  regression, classification, ...

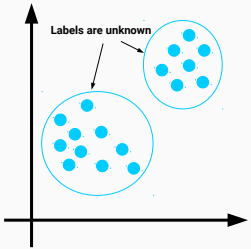


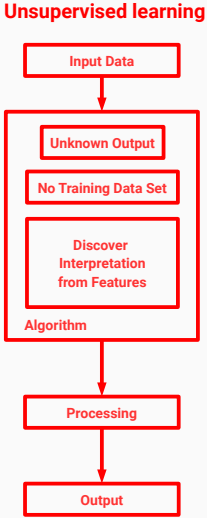


Machine learning algorithms

Machine learning algorithms:

- Supervised learning:
  regression, classification, ...
- Unsupervised learning:
  clustering, dim-reduction, ...

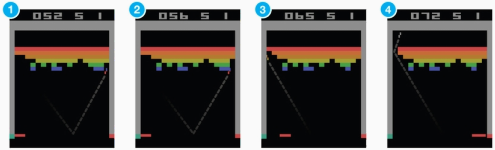




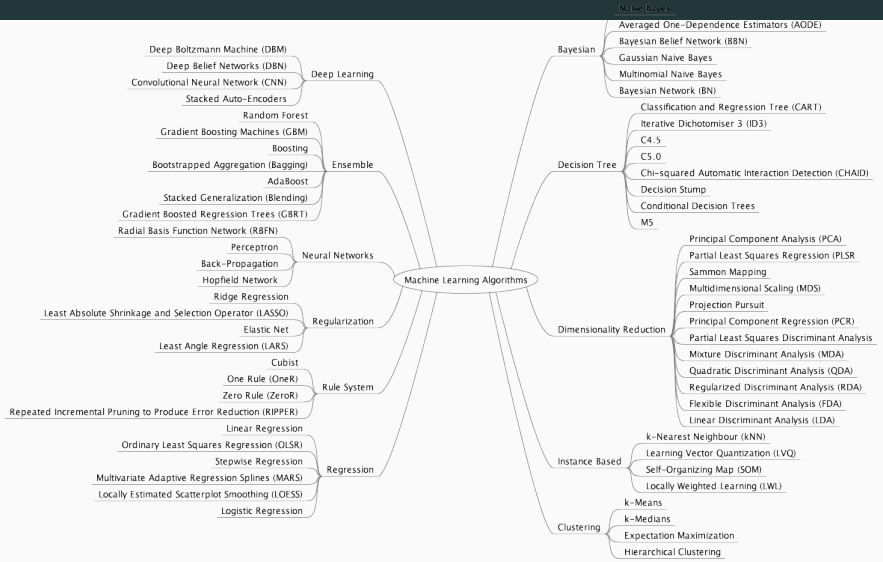
Machine learning algorithms

Machine learning algorithms:

- Supervised learning:
  regression, classification, ...
- Unsupervised learning:
  clustering, dim-reduction, ...
- Reinforcement learning:
  real-time decisions, ...



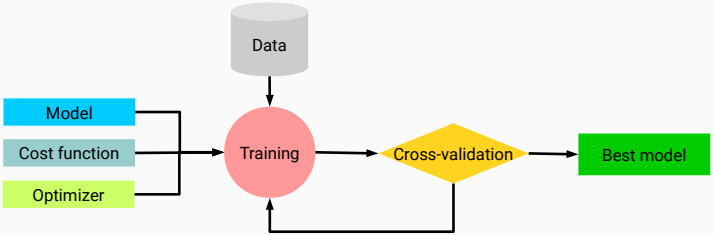
Machine learning algorithms



More than 60 algorithms.

Workflow in machine learning

The operative workflow in ML is summarized by the following steps:

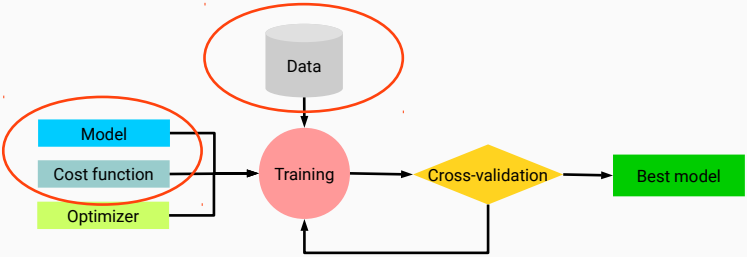


The best model is then used to:

- supervised learning: make predictions for new observed data.
- unsupervised learning: extract features from the input data.

Models and metrics

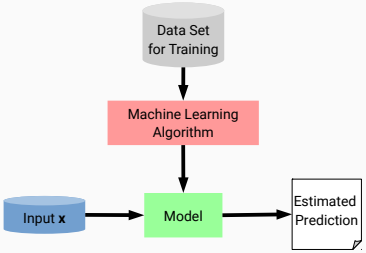
Models and metrics



Model representation in supervised learning

We define parametric and structure models for statistical inference:

- **examples:** linear models, neural networks, decision tree...



- Given a training set of input-output pairs $A = (x_1, y_1), \ldots, (x_n, y_n)$.
- Find a model $\mathcal{M}$ which:

$$\mathcal{M}(\boldsymbol{x}) \sim \boldsymbol{y}$$

where $\boldsymbol{x}$ is the input vector and $\boldsymbol{y}$ discrete labels in classification and real values in regression.

Model representation in supervised learning

Examples of models:

$\rightarrow$ linear regression we define a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output:

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters and b a constant.

Model representation in supervised learning

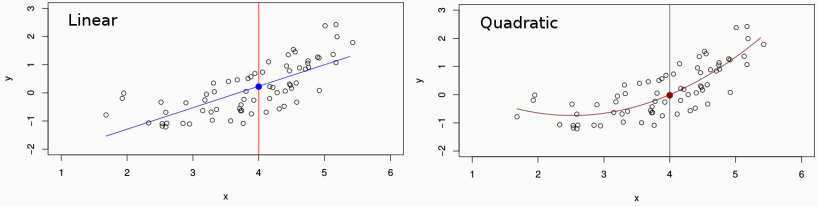
Examples of models:

→ linear regression we define a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output:

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters and b a constant.

→ Generalized linear models are also available increasing the power of linear models:



Model representation in supervised learning

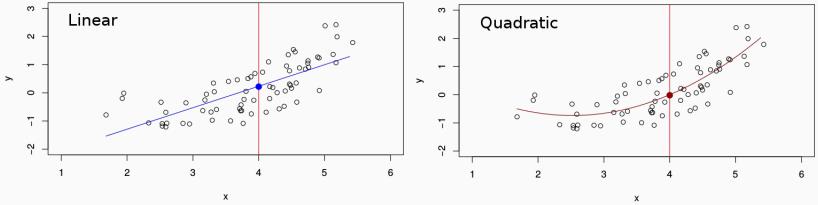
Examples of models:

→ linear regression we define a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output:

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters and b a constant.

→ Generalized linear models are also available increasing the power of linear models:

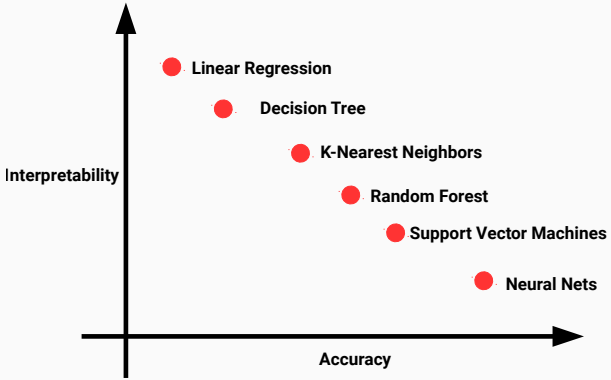


→ Non-linear models: neural networks (talk later).

Model representation trade-offs

However, the selection of the appropriate model comes with trade-offs:

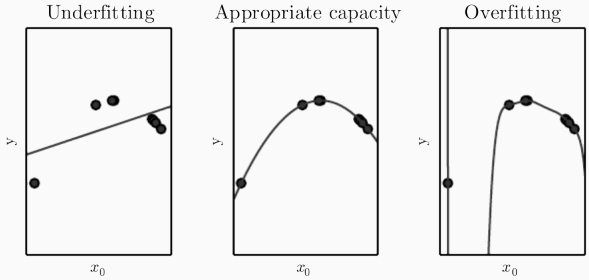
- **Prediction accuracy vs interpretability**:
  → *e.g.* linear model vs splines or neural networks.



Model representation trade-offs

However, the selection of the appropriate model comes with trade-offs:

- **Prediction accuracy vs interpretability**:
  → *e.g.* linear model vs splines or neural networks.
- **Optimal capacity/flexibility:** number of parameters, architecture
  → deal with overfitting, and underfitting situations



Assessing the model performance

How to check model performance?

→ define metrics and statistical estimators for model performance.

Examples:

- Regression: cost / loss / error function,
- Classification: cost function, precision, accuracy, recall, ROC, AUC

Assessing the model performance - cost function

To access the model performance we define a cost function $J(\boldsymbol{w})$ which often measures the difference between the target and the model output.

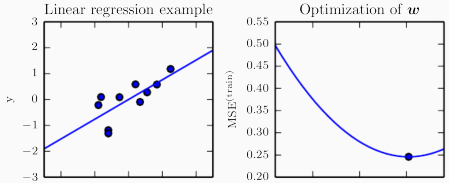
In a optimization procedure, given a model $\hat{y}_{\boldsymbol{w}}$, we search for:

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$

The mean square error (MSE) is the most commonly used for regression:

$$J(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))^2$$

a quadratic function and convex function in linear regression.



Assessing the model performance - cost function

Other cost functions are depending on the nature of the problem.

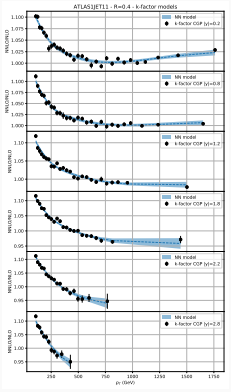
Some other examples:

- regression with uncertainties, chi-square:

$$J(\boldsymbol{w}) = \sum_{i,j=1}^{n} (y_i - \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))(\sigma^{-1})_{ij}(y_j - \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_j))$$

where:

- σ_{ij} is the data covariance matrix.
  e.g. for LHC data experimental statistical and systematics correlations.

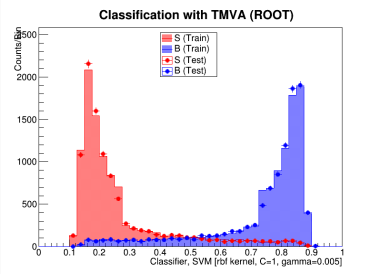


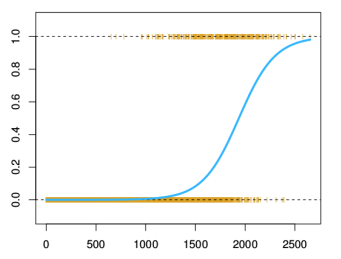
Assessing the model performance - cost function

- logistic regression (binary classification): cross-entropy

$$J(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i) + (1-y_i)\log(1-\hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))$$

where $\hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i) = 1/(1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i})$.

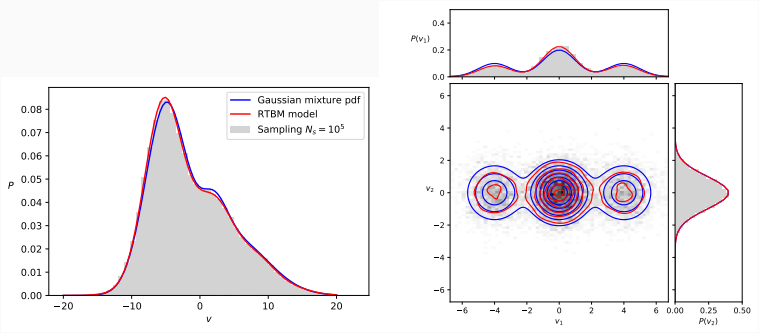




Assessing the model performance - cost function

- density estimate / regression: negative log-likelihood:

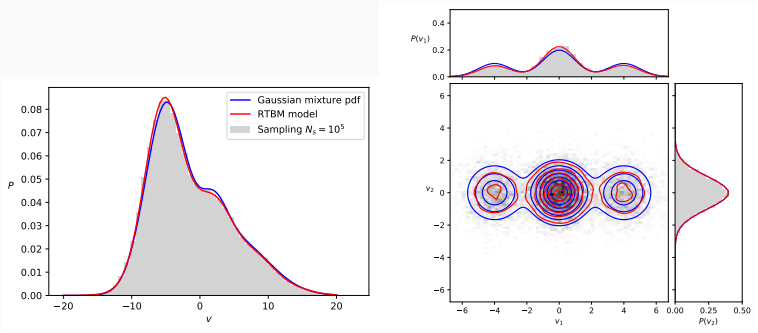
$$J(\boldsymbol{w}) = -\sum_{i=1}^{n} \log(\hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))$$



Assessing the model performance - cost function

- density estimate / regression: negative log-likelihood:

$$J(\boldsymbol{w}) = -\sum_{i=1}^{n} \log(\hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))$$



- Kullback-Leibler, RMSE, MAE, etc.

Training and test sets

Another common issue related to model capacity in supervised learning:

- The model should not learn **noise** from data.
- The model should be able to **generalize** its output to new samples.

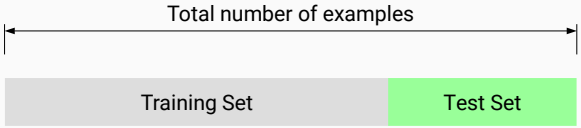
Training and test sets

Another common issue related to model capacity in supervised learning:

- The model should not learn **noise** from data.
- The model should be able to **generalize** its output to new samples.

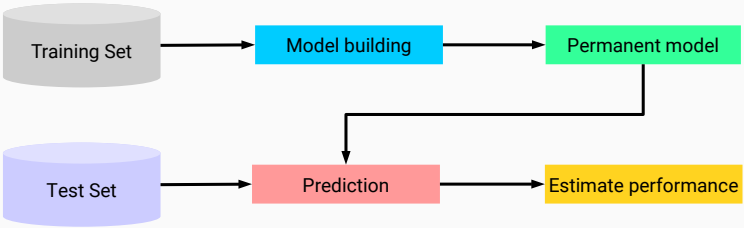
To observe this issue we split the input data in training and test sets:

- training set error, $J_{\text{Tr}}(\boldsymbol{w})$
- test set/generalization error, $J_{\text{Test}}(\boldsymbol{w})$



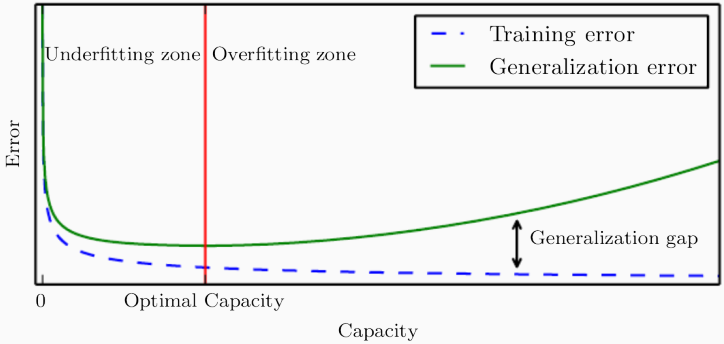
Training and test sets

The test set is independent from the training set but follows the same probability distribution.



Bias-variance trade-off

From a practical point of view dividing the input data in training and test:



The training and test/generalization error conflict is known as bias-variance trade-off.

Bias-variance trade-off

Supposing we have model $\hat{y}(\boldsymbol{x})$ determined from a training data set, and considering as the true model

$$Y = y(X) + \epsilon, \text{ with } y(x) = E(Y|X = x),$$

where the noise ϵ has zero mean and constant variance.

Bias-variance trade-off

Supposing we have model $\hat{y}(\boldsymbol{x})$ determined from a training data set, and considering as the true model

$$Y = y(X) + \epsilon, \text{ with } y(x) = E(Y|X = x),$$

where the noise ϵ has zero mean and constant variance.

If we take (x_0, y_0) from the test set then:

$$E[(y_0 - \hat{y}(x_0))^2] = (\text{Bias}[\hat{y}(x_0)])^2 + \text{Var}[\hat{y}(x_0)] + \text{Var}(\epsilon),$$

where

- $\text{Bias}[\hat{y}(x_0)] = E[\hat{y}(x_0)] - y(x_0)$
- $\text{Var}[\hat{y}(x_0)] = E[\hat{y}(x_0)^2] - (E[\hat{y}(x_0)])^2$

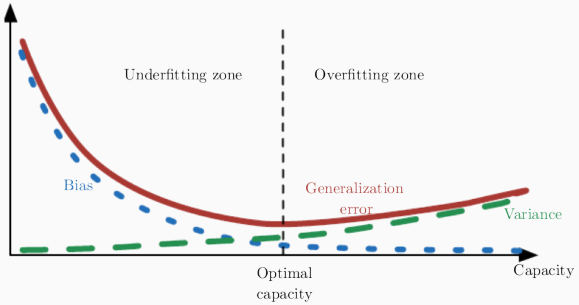
So, the expectation averages over the variability of y_0 (bias) and the variability in the training data.

Bias-variance trade-off

If $\hat{y}$ increases **flexibility**, its variance increases and its biases decreases.

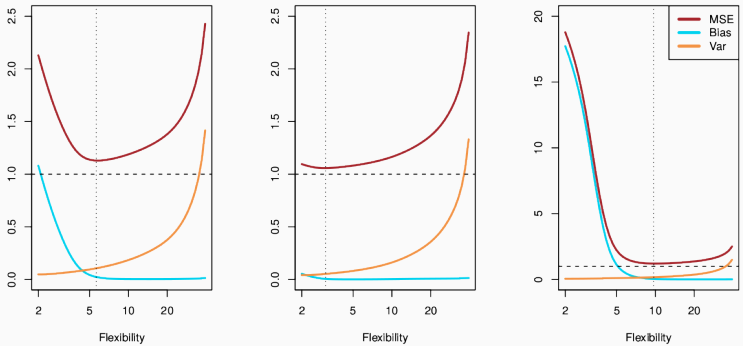
Choosing the flexibility based on average test error amounts to a bias-variance trade-off:

- **High Bias** $\rightarrow$ underfitting:
  erroneous assumptions in the learning algorithm.
- **High Variance** $\rightarrow$ overfitting:
  erroneous sensitivity to small fluctuations (noise) in the training set.



Bias-variance trade-off

More examples of bias-variance trade-off:



Bias-variance trade off

Regularization techniques can be applied to modify the learning algorithm and reduce its generalization error but not its training error.

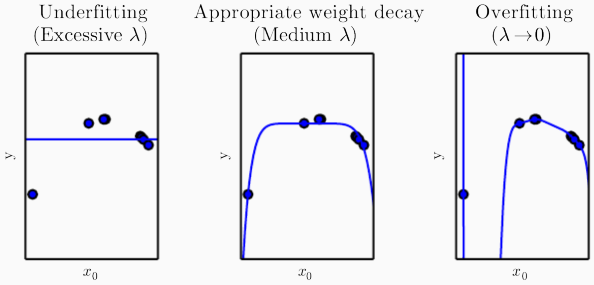
For example, including the weight decay to the MSE cost function:

$$J(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 + \lambda \boldsymbol{w}^T \boldsymbol{w}.$$

where λ is a real number which express the preference for weights with smaller squared L^2 norm.

Solution for the bias-variance trade off

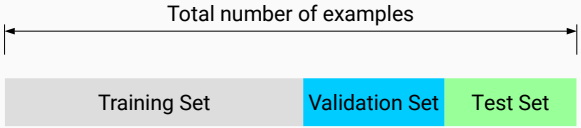
Tuning the hyperparameter λ we can regularize a model without modifying explicitly its capacity.



Solution for the bias-variance trade off

A common way to reduce the bias-variance trade-off and choose the proper learning hyperparamters is to create a validation set that:

- not used by the training algorithm
- not used as test set

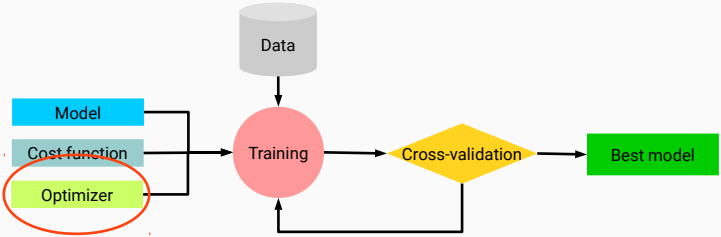


- **Training set**: examples used for learning.
- **Validation set**: examples used to tune the hyperparameters.
- **Test set**: examples used only to access the performance.

Techniques are available to deal with data samples with large and small number of examples. (talk later)

Parameter learning

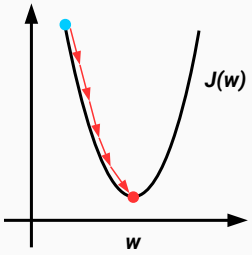
Parameter learning



Cost function minimization

Optimization algorithms **minimize an objective function**, $J(\boldsymbol{w})$, that depends on the model internal learnable parameters $\boldsymbol{w}$.

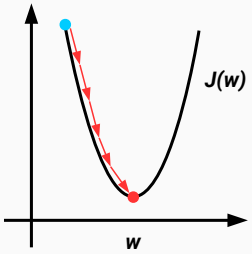
$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$



Cost function minimization

Optimization algorithms **minimize an objective function**, $J(\boldsymbol{w})$, that depends on the model internal learnable parameters $\boldsymbol{w}$.

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$



The most popular techniques are:

- normal equations (least squares)
- derivative based optimization
- evolutionary algorithms

The choice of a technique depends on the model and problem employed.

Normal equations

The normal equation is a method to solve for $\boldsymbol{w}$ analytically.

- it is employed in linear and non-linear least squares optimization.
- it is fast for small models with few features, otherwise it can be computationally intensive and slow.

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

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

How to solve it when the system if overdetermined?

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

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

How to solve it when the system if overdetermined?

1. Define the cost function for the problem:

$$J(\boldsymbol{w}) = ||\boldsymbol{y} - X\boldsymbol{w}||^2$$

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

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

How to solve it when the system if overdetermined?

1. Define the cost function for the problem:

$$J(\boldsymbol{w}) = ||\boldsymbol{y} - X\boldsymbol{w}||^2$$

2. Compute and impose:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \mathbf{0}$$

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

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

How to solve it when the system if overdetermined?

1. Define the cost function for the problem:

$$J(\boldsymbol{w}) = ||\boldsymbol{y} - X\boldsymbol{w}||^2$$

2. Compute and impose:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \mathbf{0}$$

3. We obtain the solution:

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

with X^+ is the pseudoinverse of X.

Derivative based optimization

More general optimization algorithms based on derivatives:

- **First order optimization algorithms:** uses the gradient of the cost function with respect to the parameters in a iteractive procedure.
  $\rightarrow$ gradient descent algorithms.

Derivative based optimization

More general optimization algorithms based on derivatives:

- **First order optimization algorithms:** uses the gradient of the cost function with respect to the parameters in a iteractive procedure.
  $\rightarrow$ gradient descent algorithms.
- **Second order optimization algorithms:** uses the Hessian of the cost function and takes care of the curvature of surface.
  $\rightarrow$ if the Hessian is known it may be faster than gradient descent,
  $\rightarrow$ otherwise slow due to the Hessian evaluation.

Derivative based optimization

More general optimization algorithms based on derivatives:

- **First order optimization algorithms:** uses the gradient of the cost function with respect to the parameters in a iteractive procedure.
  $\rightarrow$ gradient descent algorithms.
- **Second order optimization algorithms:** uses the Hessian of the cost function and takes care of the curvature of surface.
  $\rightarrow$ if the Hessian is known it may be faster than gradient descent,
  $\rightarrow$ otherwise slow due to the Hessian evaluation.

In practice Gradient Descent is the most popular technique in ML.

Which method?

Q: normal equation or derivative based?

Suppose we have m training examples and n features.

Normal equation

- ✓ no parameters to tune
- ✓ no iterations
- ✗ slow if n is large
- ✗ requires $(X^T X)^{-1}, \mathcal{O}(n^3)$

Which method?

Q: normal equation or derivative based?

Suppose we have m training examples and n features.

Normal equation	Gradient descent
✓ no parameters to tune	✓ efficient when n is large
✓ no iterations	✓ easy to implement/use
✗ slow if n is large	✗ requires iterations
✗ requires $(X^T X)^{-1}, \mathcal{O}(n^3)$	✗ requires parameter tune

Gradient descent idea

Basic idea:
Assuming we want to minimize $J(\boldsymbol{w})$ where $\boldsymbol{w}$ is a vector of parameters:

- select a initial solution vector $\boldsymbol{w}$,
- change the $\boldsymbol{w}$ to reduce $J(\boldsymbol{w})$

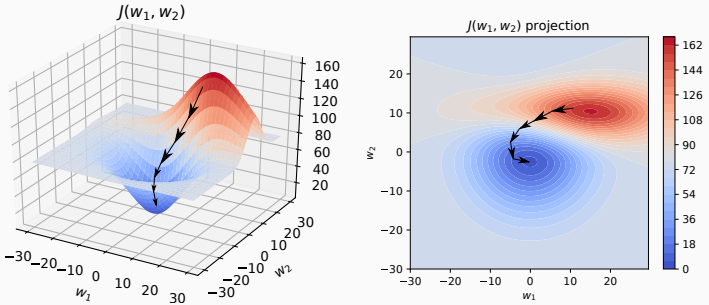
Repeat until a minimum of $J(\boldsymbol{w})$ is reached.

Gradient descent idea

Basic idea:
Assuming we want to minimize $J(\boldsymbol{w})$ where $\boldsymbol{w}$ is a vector of parameters:

- select a initial solution vector $\boldsymbol{w}$,
- change the $\boldsymbol{w}$ to reduce $J(\boldsymbol{w})$

Repeat until a minimum of $J(\boldsymbol{w})$ is reached.



Gradient descent algorithm

Simultaneously for each parameter in $\boldsymbol{w}$ repeat until convergence:

$$w_i := w_i - \eta \frac{\partial}{\partial w_i} J(\boldsymbol{w})$$

where η is the learning rate, $\eta \geq 0$, it can be a fixed number, because the gradient term will automatically compensate with smaller steps: $\nabla_w J|_{w \to w_{\text{best}}} \to 0$.

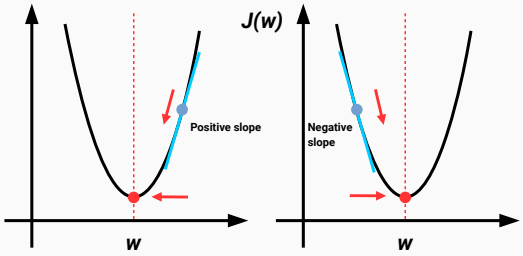
Gradient descent algorithm

Simultaneously for each parameter in $\boldsymbol{w}$ repeat until convergence:

$$w_i := w_i - \eta \frac{\partial}{\partial w_i} J(\boldsymbol{w})$$

where η is the learning rate, $\eta \geq 0$, it can be a fixed number, because the gradient term will automatically compensate with smaller steps: $\nabla_w J|_{w \to w_{\text{best}}} \to 0$.

Why the negative sign in term $-\eta$? (example in 1D)

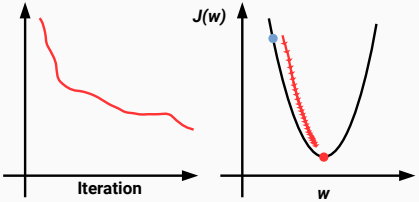


- if $\nabla_w J(w) > 0$ then w decreases
- if $\nabla_w J(w) < 0$ then w increases

Gradient descent and learning rate

The η is another example of hyperparameter which requires tune.

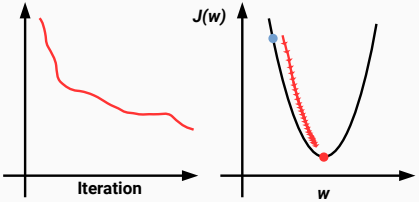
- if η is **too small**, gradient descent can be slow.



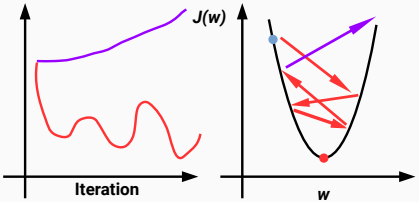
Gradient descent and learning rate

The η is another example of hyperparameter which requires tune.

- if η is **too small**, gradient descent can be slow.



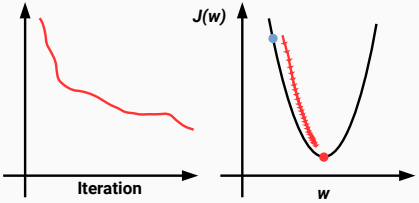
- if η is **too large**, gradient descent may fail to converge or diverge.



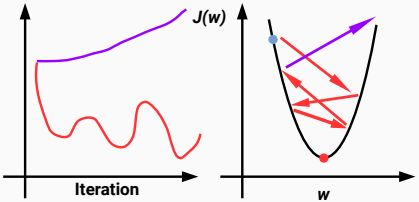
Gradient descent and learning rate

The η is another example of hyperparameter which requires tune.

- if η is **too small**, gradient descent can be slow.



- if η is **too large**, gradient descent may fail to converge or diverge.



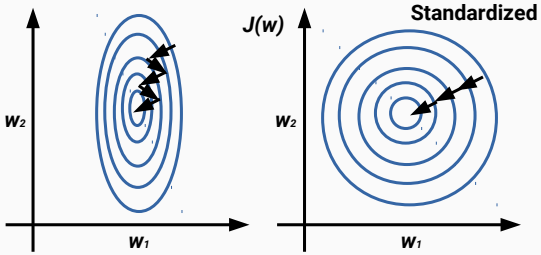
Gradient descent and feature scaling

Another practical hint: **feature scaling**.

Make sure the input features x_i are in a similar scale, *e.g.* standardization:

$$x_i := \frac{x_i - \mu_{x_i}}{\sigma_{x_i}}$$

where μ_{x_i} and σ_{x_i} are the mean and standard deviation respectively.



Gradient descent variants

When performing gradient descent the cost function $J(\boldsymbol{w})$ is evaluated over the training data, *e.g.* for the MSE cost function:

$$\frac{\partial}{\partial \boldsymbol{w}} J(\boldsymbol{w}) = \frac{\partial}{\partial \boldsymbol{w}} \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 \right).$$

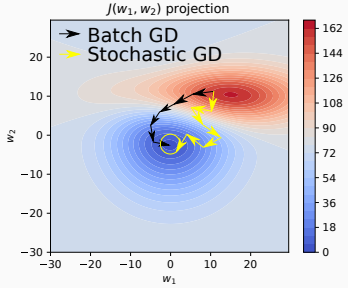
If the training data set is too large, there are gradient descent variations that may improve convergence in terms of speed and quality:

- Batch Gradient Descent: all training data points are evaluated in the cost function gradient at each iteration.

Gradient descent variants

- Stochastic Gradient Descent (SGD):
  1. randomly shuffle training examples,
  2. use 1 example at each iteration.

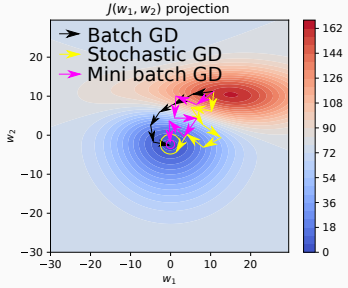
  Features:
  - parameters updates have high variance and cost function fluctuates.
  - helps to discover new and possibly better minima.
  - requires to slowly decrease the learning rate η to reduce fluctuations.



Gradient descent variants

- Mini Batch Gradient Descent:
  1. use a subset of size b (batch size) of examples in each iteration,
  2. use the batch set example at each iteration.

  Features:
  - takes the best from both previous methods,
  - reduces the variance in the parameter updates (stable convergence),
  - good for data parallelism, efficient for matrix operations



Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

- Root Mean Square Propagation (RMSProp): it introduces an adaptive learning rate for each parameter:

$$\boldsymbol{w} := \boldsymbol{w} - \frac{\eta}{\sqrt{v(\boldsymbol{w}, ite)}} \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

- Root Mean Square Propagation (RMSProp): it introduces an adaptive learning rate for each parameter:

$$\boldsymbol{w} := \boldsymbol{w} - \frac{\eta}{\sqrt{v(\boldsymbol{w}, ite)}} \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

- Others: Averaging, AdaGrad, Adam, etc...

Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

- Root Mean Square Propagation (RMSProp): it introduces an adaptive learning rate for each parameter:

$$\boldsymbol{w} := \boldsymbol{w} - \frac{\eta}{\sqrt{v(\boldsymbol{w}, ite)}} \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

- Others: Averaging, AdaGrad, Adam, etc...

All these schemes and respective parameters can be considered as extra hyperparameters to tune.

Examples of second order optimization

Popular examples of second order optimization algorithms:

- Newton's method: an iterative method based on Taylor expansion.
  Example in 1D: consider the Taylor expansion

$$J_T(w) = J_T(w_n + \Delta w) \approx J(w_n) + J'(w_n)\Delta w + \frac{1}{2}J''(w_n)\Delta w^2$$

  We aim to find Δw which satisfies:

$$\nabla_{\Delta w} J_T(w_n + \Delta w) = J'(w_n) + J''(w_n)\Delta w = 0$$

$$w_{n+1} = w_n + \Delta w = w_n - \frac{J'(w_n)}{J''(w_n)}$$

  $w_1, w_2, \ldots$ will converge to a stationary point w^* where $J'(w^*) = 0$.

Examples of second order optimization

Popular examples of second order optimization algorithms:

- Newton's method: an iterative method based on Taylor expansion.
  Example in 1D: consider the Taylor expansion

$$J_T(w) = J_T(w_n + \Delta w) \approx J(w_n) + J'(w_n)\Delta w + \frac{1}{2}J''(w_n)\Delta w^2$$

  We aim to find Δw which satisfies:

$$\nabla_{\Delta w} J_T(w_n + \Delta w) = J'(w_n) + J''(w_n)\Delta w = 0$$

$$w_{n+1} = w_n + \Delta w = w_n - \frac{J'(w_n)}{J''(w_n)}$$

  $w_1, w_2, \ldots$ will converge to a stationary point w^* where $J'(w^*) = 0$.
  Generalization in N dimensions:

$$(\boldsymbol{H}J(\boldsymbol{w_n}))\boldsymbol{\Delta w} = -\nabla J(\boldsymbol{w}_n)$$

  where $\boldsymbol{H}$ is the Hessian matrix.

Examples of second order optimization

Popular examples of second order optimization algorithms:

- Quasi-newton methods: *i.e.* methods which optimizes even if the Hessian matrix is expensive or not available. The Taylor's series is:

$$J(\boldsymbol{w}_n + \Delta\boldsymbol{w}) \approx J(\boldsymbol{w}_n) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n)^T \Delta\boldsymbol{w} + \frac{1}{2}\Delta\boldsymbol{w}^T B \Delta\boldsymbol{w},$$

  where B is an approximation to the Hessian matrix and

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n + \Delta\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n) + B\Delta\boldsymbol{w},$$

  which produces the Newton step:

$$\Delta\boldsymbol{w} = -B^{-1}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n).$$

  Some methods: BFGS, L-BFGS, DFP, Broyden.
  - differ by the choice of the solution to update B.

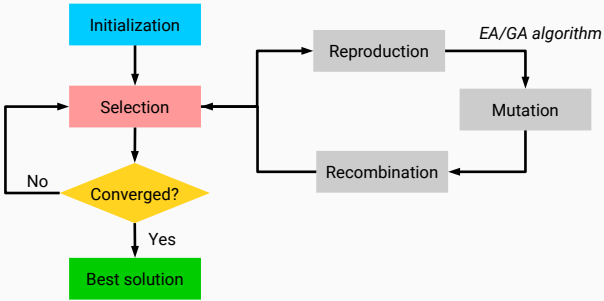
Popular in ML since the beginning of the deep learning era.

Evolutionary algorithms

Evolutionary algorithms (EA), inspired by biological evolution, is a generic population-based metaheuristic optimization algorithm.

Techniques in EA use mechanisms such as: reproduction, mutation, recombination, and selection.

Genetic algorithm is the most popular technique of EA.

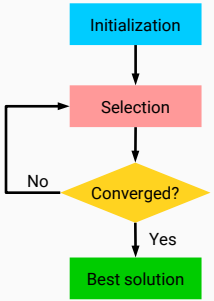


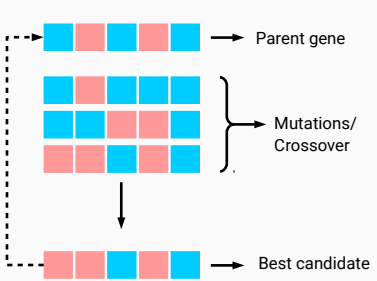
Genetic algorithm

Genetic algorithm is well suited when:

- gradients are not available,
- non parametric function,
- non homogeneous cost function along all training set points.

Example:





Questions?