Lectures on Machine Learning

Lecture 3: practical aspects of machine learning

Stefano Carrazza

TAE2018, 2-15 September 2018

European Organization for Nuclear Research (CERN)

Acknowledgement: This project has received funding from HICCUP ERC Consolidator grant (614577) and by the European Unions Horizon 2020 research and innovation programme under grant agreement no. 740006.

Machine Learning • PDFs • QCD

Outline

Lecture 1 (yesterday)

- Artificial intelligence
- Machine learning
- Model representation
- Metrics

Lecture 2 (today)

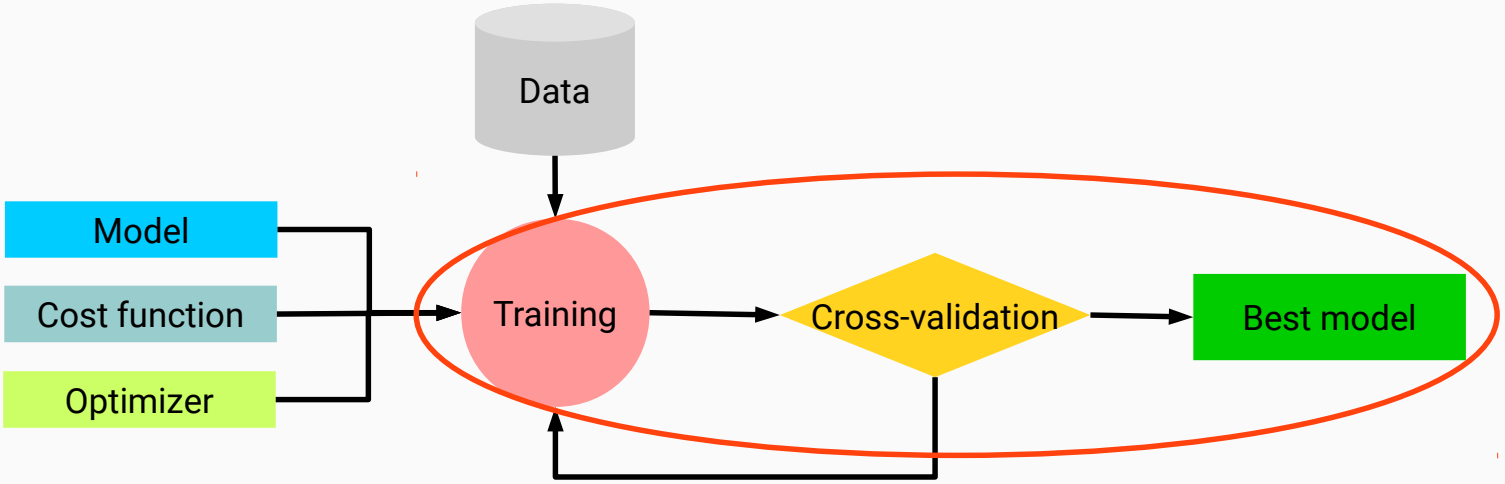
- Parameter learning
- Non-linear models
- Beyond neural networks
- Clustering

Lecture 3 (today)

- Hyperparameter tune
- Cross-validation
- ML in practice
- The PDF case study

Hyperparameter tune

Outline


Data
Model
Cost function
Optimizer
Training
Cross-validation
Best model

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions
- Regularization techniques (to avoid overfitting):

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions
- Regularization techniques (to avoid overfitting):
  - weight decay (parameter λ)

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions
- Regularization techniques (to avoid overfitting):
  - weight decay (parameter λ)
  - if SGD: early stopping techniques

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions
- Regularization techniques (to avoid overfitting):
  - weight decay (parameter λ)
  - if SGD: early stopping techniques
  - if NN: dropout, artificial data, stochastic pooling, etc...

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions
- Regularization techniques (to avoid overfitting):
  - weight decay (parameter λ)
  - if SGD: early stopping techniques
  - if NN: dropout, artificial data, stochastic pooling, etc...
- Training:

Hyperparameters summary

So far we have encountered the following hyperparameters:

- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions
- Regularization techniques (to avoid overfitting):
  - weight decay (parameter λ)
  - if SGD: early stopping techniques
  - if NN: dropout, artificial data, stochastic pooling, etc...
- Training:
  - the SGD learning parameters η

Hyperparameters summary

So far we have encountered the following hyperparameters:

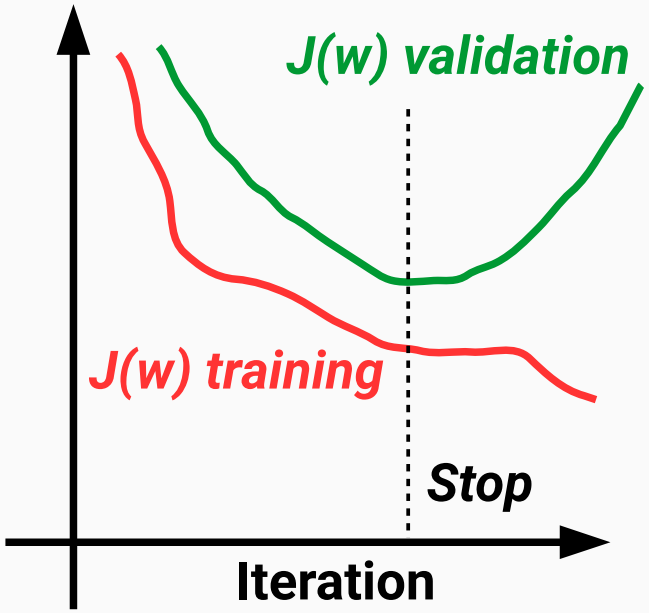
- Model related:
  - model architecture / size
  - if NN: layers, nodes, activation functions
- Regularization techniques (to avoid overfitting):
  - weight decay (parameter λ)
  - if SGD: early stopping techniques
  - if NN: dropout, artificial data, stochastic pooling, etc...
- Training:
  - the SGD learning parameters η
  - other SGD parameters depending on the gradient descent scheme

Other regularization techniques

Example: early stopping techniques are applied when the optimization is performed in an iterative procedure, *.e.g.* via gradient descent.

Other regularization techniques

Example: early stopping techniques are applied when the optimization is performed in an iterative procedure, *.e.g.* via gradient descent.



These techniques monitor the cost function for the validation set and stop when this quantity has stopped improving:

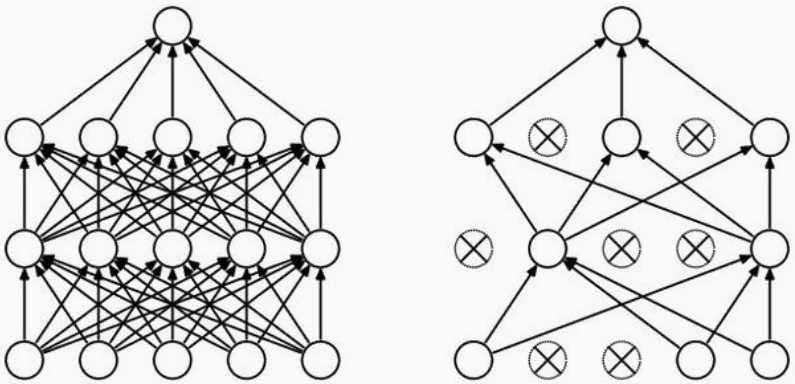
- look at the variation in a moving window
- stop at the minimum of the validation set (lookback method),

Other regularization techniques

Example: neural network dropout

At each training stage:

- individual nodes and related incoming and outgoing edges are dropped-out of the neural network with a fixed probability.
- the reduced NN is trained on the data.
- the removed nodes are reinserted in the NN with their original weights.



(a) Standard Neural Net (b) After applying dropout.

Other regularization techniques

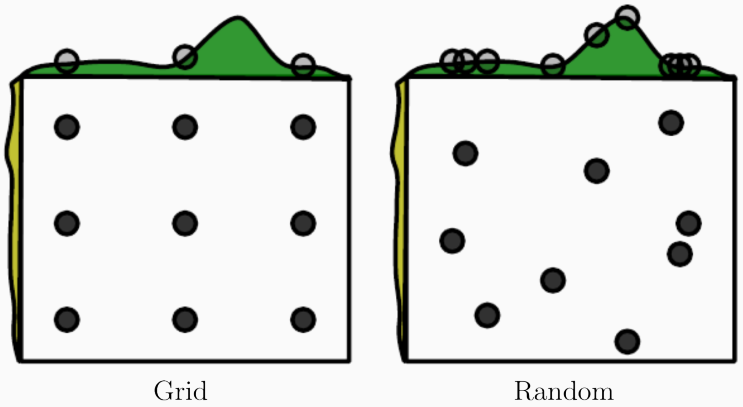
How should we proceed with hyperparameter tune?

Other regularization techniques

How should we proceed with hyperparameter tune?

Possible solutions:

- grid search: exhaustive searching through a manually subset range of hyperparameter space
- random search: specially powerful with small number of hyperparameters affects the final performance of the ML algorithm.

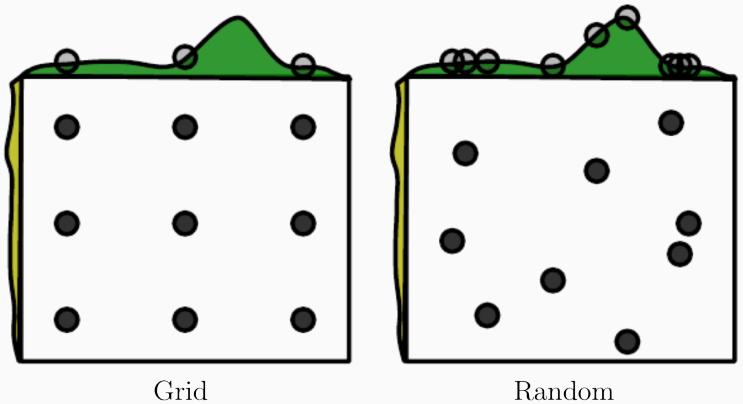


Other regularization techniques

How should we proceed with hyperparameter tune?

Possible solutions:

- grid search: exhaustive searching through a manually subset range of hyperparameter space
- random search: specially powerful with small number of hyperparameters affects the final performance of the ML algorithm.



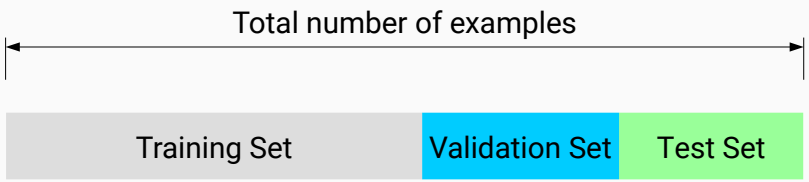
Other useful methods:

- bayesian optimization
- gradient-based optimization
- evolutionary optimization

Cross-validation

Cross-validation

The hyperparameter tune procedure still requires the training/validation/test split to choose for the best model.

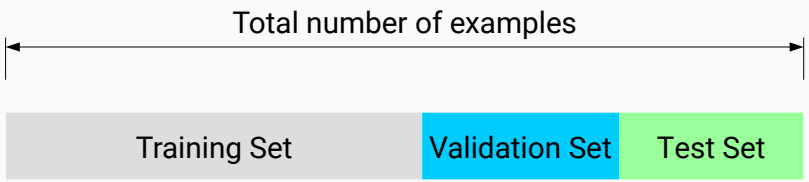


Problems:

- how to perform the data split when the available data set is small?
- how to define a suitable split?

Cross-validation

The hyperparameter tune procedure still requires the training/validation/test split to choose for the best model.



Problems:

- how to perform the data split when the available data set is small?
- how to define a suitable split?

Solution:

Use cross-validation algorithms to access the quality of your model + hyperparameter choice.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
  - Leave-p-out: uses p observations as validation set.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
  - Leave-p-out: uses p observations as validation set.
  - Leave-one-out: set $p = 1$.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
  - Leave-p-out: uses p observations as validation set.
  - Leave-one-out: set $p = 1$.
- Non-exhaustive cross-validation: do not test all possible ways to divide the original sample but use discrete subsamples.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

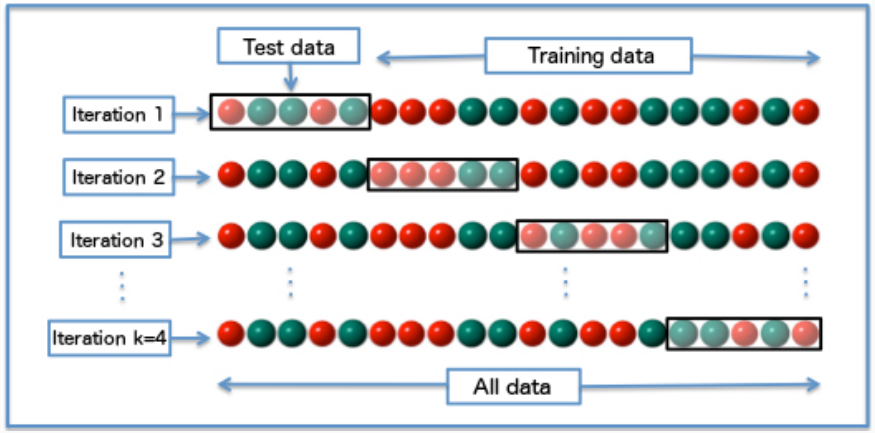
- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
  - Leave-p-out: uses p observations as validation set.
  - Leave-one-out: set $p = 1$.
- Non-exhaustive cross-validation: do not test all possible ways to divide the original sample but use discrete subsamples.
  - k-fold cross-validation.

Example k-fold cross-validation

k-fold cross-validation:

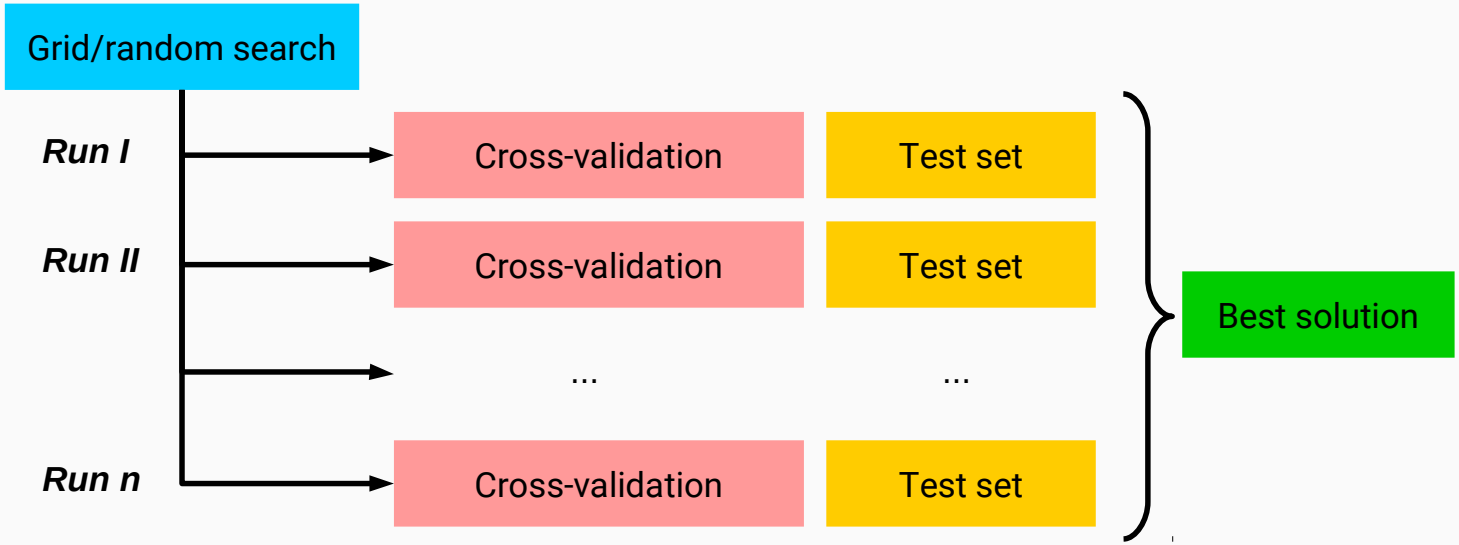
1. the original data is randomly partitioned into k equal sized subsamples.
2. from the k subsamples, a single subsample is used as validation data and the remaining $k-1$ subsamples are used as training data.
3. repeat the process k times by changing the validation and training partitions.
4. compute the average over the k results.

Example of k-fold with $k = 4$:



Complete recipe

Perform hyperparameter tune coupled to cross-validation:

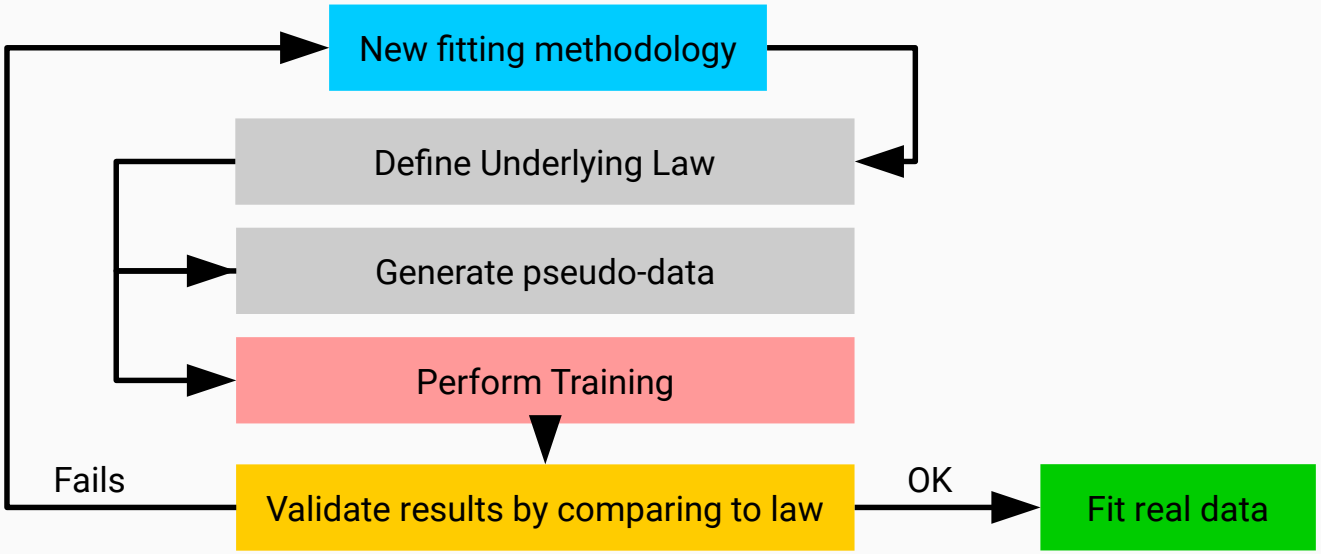


Easy parallelization at search and cross-validation stages.

Closure testing

Closure tests

Validation and optimization of fitting strategy performed on **closure test** with known underlying law.



ML in practice

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

For ML applications:

- Keras: a Python deep learning library.
- Theano: a Python library for optimization.
- PyTorch: a DL framework for fast, flexible experimentation.
- Caffe: speed oriented deep learning framework.
- MXNet: deep learning frameowrk for neural networks.
- CNTK: Microsoft Cognitive Toolkit.

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

For ML applications:

- Keras: a Python deep learning library.
- Theano: a Python library for optimization.
- PyTorch: a DL framework for fast, flexible experimentation.
- Caffe: speed oriented deep learning framework.
- MXNet: deep learning frameowrk for neural networks.
- CNTK: Microsoft Cognitive Toolkit.

For ML and beyond:

- TensorFlow: libray for numerical computation with data flow graphs.
- scikit-learn: general machine learning package.

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

For ML applications:

- Keras: a Python deep learning library.
- Theano: a Python library for optimization.
- PyTorch: a DL framework for fast, flexible experimentation.
- Caffe: speed oriented deep learning framework.
- MXNet: deep learning frameowrk for neural networks.
- CNTK: Microsoft Cognitive Toolkit.

For ML and beyond:

- TensorFlow: libray for numerical computation with data flow graphs.
- scikit-learn: general machine learning package.

Why use public codes? $\rightarrow$ builtin models and automatic differentiation

Keras

Keras is a high-level deep learning framework in Python which runs on top of TensorFlow, CNTK or Theano.

Keras

Keras is a high-level deep learning framework in Python which runs on top of TensorFlow, CNTK or Theano.

Pros:

- fast prototyping, user friendly, common code for multiple backends.
- support several NN architectures out-of-the-box.
- runs seamlessly on CPU and GPU.

Keras

Keras is a high-level deep learning framework in Python which runs on top of TensorFlow, CNTK or Theano.

Pros:

- fast prototyping, user friendly, common code for multiple backends.
- support several NN architectures out-of-the-box.
- runs seamlessly on CPU and GPU.

Cons:

- more tricky to extend when custom ML setups are required
- runs only in Python

Example of code using Keras:

```
model = Sequential() # allocate an empty model (MLP)

# append feed-forward layers 2-5-3-1
model.add(Dense(units=5, activation='sigmoid', input_dim=2))
model.add(Dense(units=3, activation='sigmoid', input_dim=5))
model.add(Dense(units=1, activation='linear', input_dim=3))

model.compile(loss='mse', optimizer='sgd') # compile the model

# train the model
model.fit(x_train, y_train, epochs=1000, batch_size=32)

# measure performance
loss_and_metrics = model.evaluate(x_test, y_test)

# generate predictions
classes = model.predict(x_test)
```

TensorFlow

TensorFlow is a library for high performance numerical computation.

TensorFlow

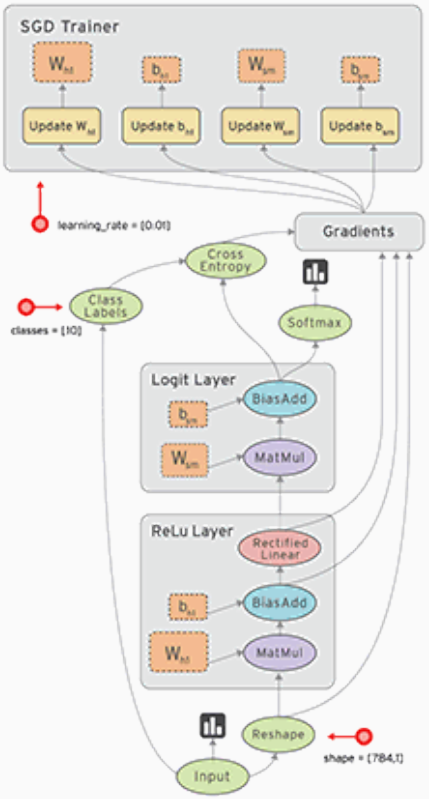
TensorFlow is a library for high performance numerical computation.

Pros:

- solves optimization problems with automatic differentiation.
- can be extended in python and c/c++.
- runs seamlessly on CPU and GPU, and can uses JIT technology.

Cons:

- do not provides builtin models from the core framework
- less automation for cross-validation and hyperparameter tune



Example of code using TensorFlow:

```
n_intput = 2
n_output = 1
n_hidden_1 = 5
n_hidden_2 = 3

# tf Graph input
X = tf.placeholder("float", [None, n_input])
Y = tf.placeholder("float", [None, n_output])

# Store layers weight & bias
weights = {
  'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
  'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
  'out': tf.Variable(tf.random_normal([n_hidden_2, n_output]))
}
biases = {
  'b1': tf.Variable(tf.random_normal([n_hidden_1])),
  'b2': tf.Variable(tf.random_normal([n_hidden_2])),
  'out': tf.Variable(tf.random_normal([n_output]))
}
```

Example of code using TensorFlow:

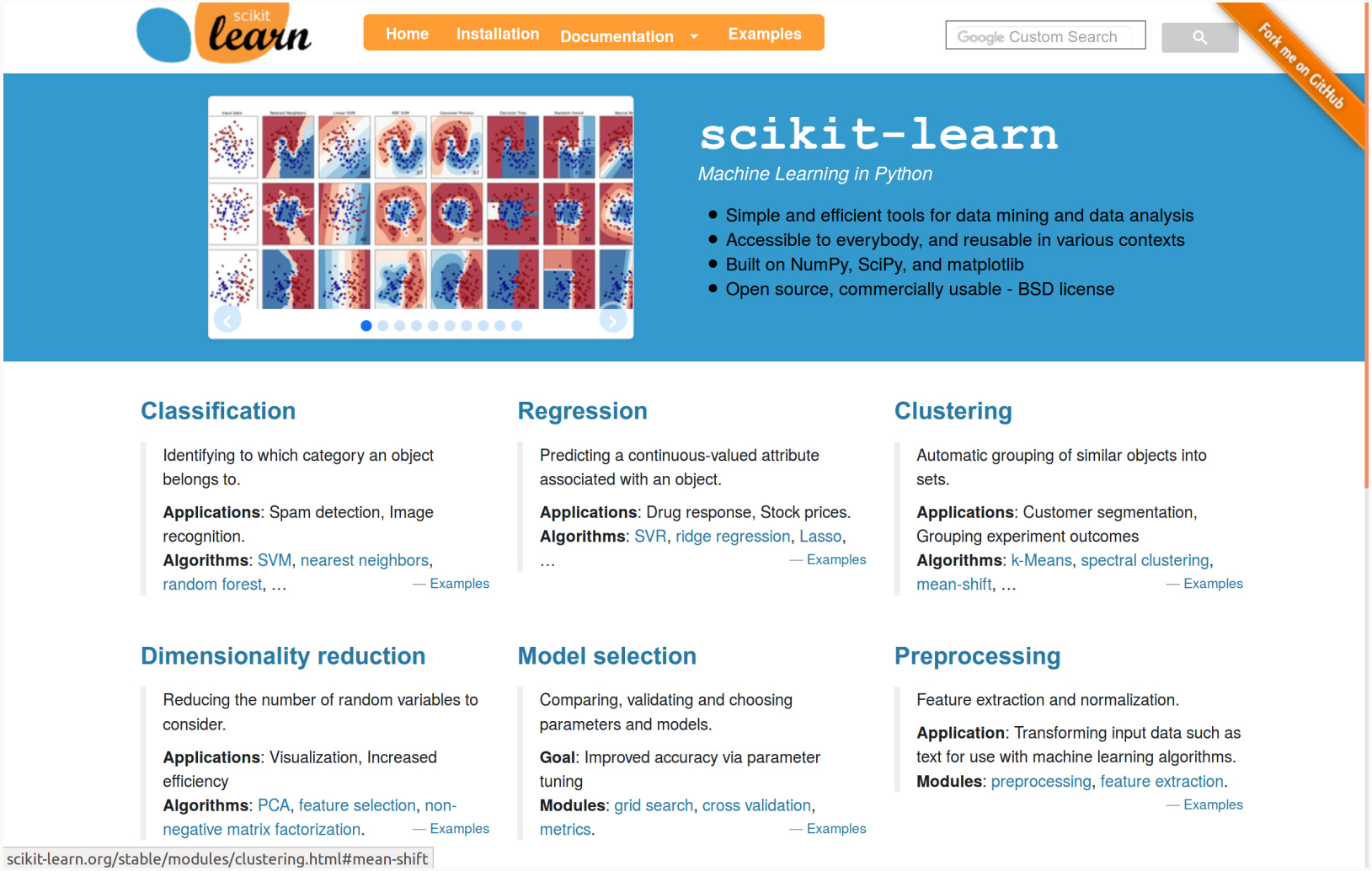
```
...

def MLP(x): # define the neural network
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    return tf.matmul(layer_2, weights['out']) + biases['out']

model = MLP(X) # attach model to the input placeholder
loss = tf.reduce_mean(tf.square(model-Y)) # evaluate loss graph
train = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

# perform training loop manually
...
for epoch in range(1000):
   _, cost = sess.run([train, loss], feed_dict={X: x_train, Y: y_train})
```

Scikit-learn



Scikit-learn

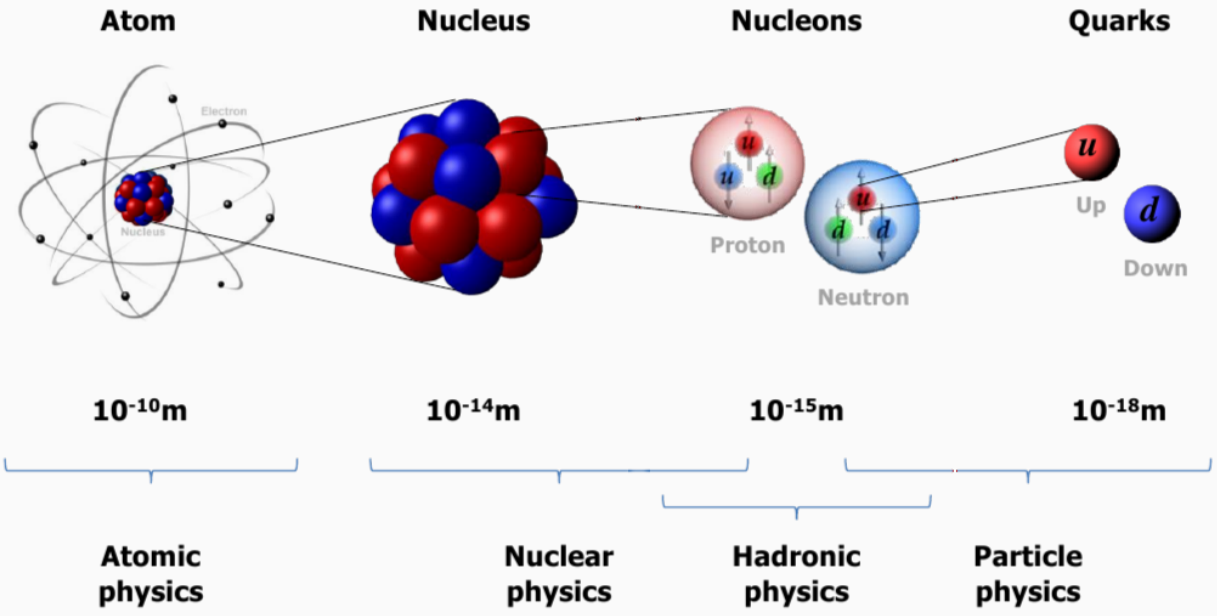
Scikit-learn contains the most popular algorithms for:

- Supervised learning: neural networks, decision trees, etc.
- Unsupervised learning: density estimate, clustering, etc.
- Model selection: cross-validation, hyperparameter tune, etc.
- Dataset transformations: feature extractions, dim. reduction, etc.
- Dataset loading
- Strategies to scale computationally
- Computational performance

The PDF case study

Parton density functions

The **parton** model was introduced by Feynman in 1969 in order to characterize **hadrons** (*e.g.* protons and neutrons) in QCD processes and interactions in high energy particle collisions.



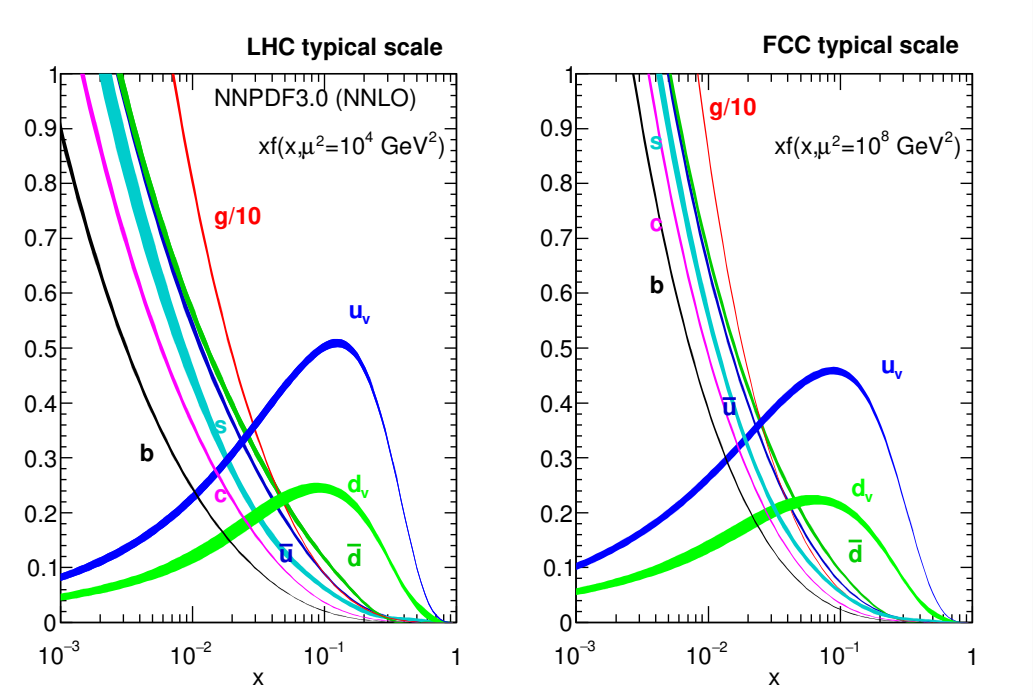
Partons are quarks and gluons characterized by a probability density functions of its nucleon momentum.

Parton density functions

- **PDFs** are **essential** for a **realistic computation** of any particle physics **observable**, σ, thanks to the factorization theorem

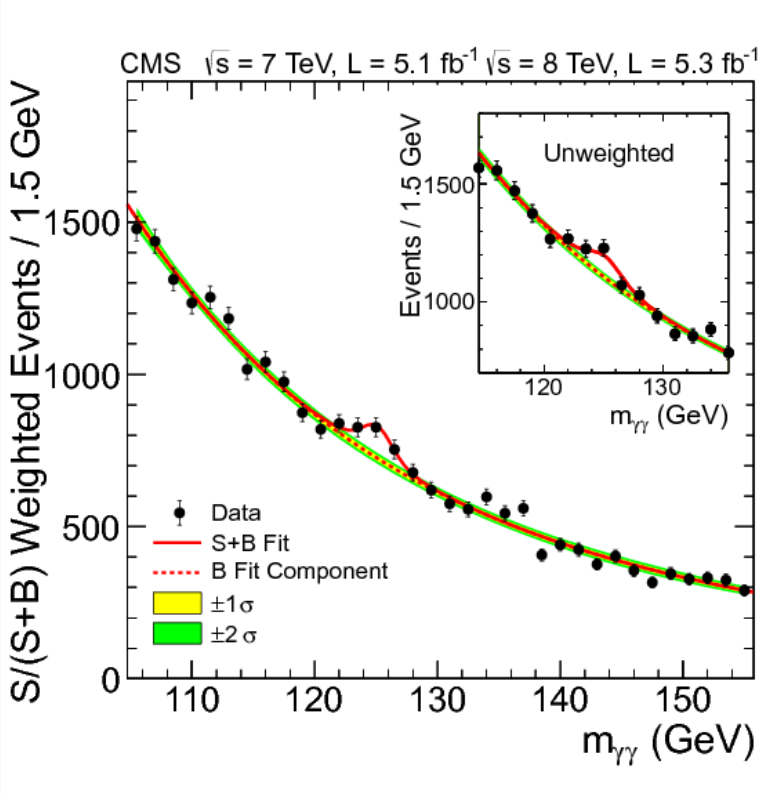
$$\sigma = \hat{\sigma} \otimes f,$$

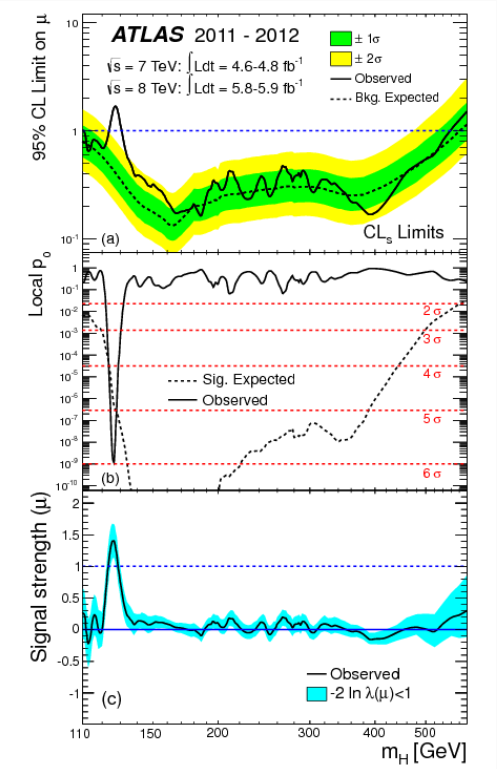
  where the elementary **hard cross-section** $\hat{\sigma}$ is convoluted with f the **PDF**.
- **PDFs** are **not calculable**: reflect non-perturbative physics of confinement.
- **PDFs** are **extracted** by comparing theoretical predictions to real data.



Parton density functions

- PDFs are **necessary** to determine theoretical predictions for **signal**/**background** of experimental **measurements**.
  - *e.g. the Higgs discovery at the LHC*:

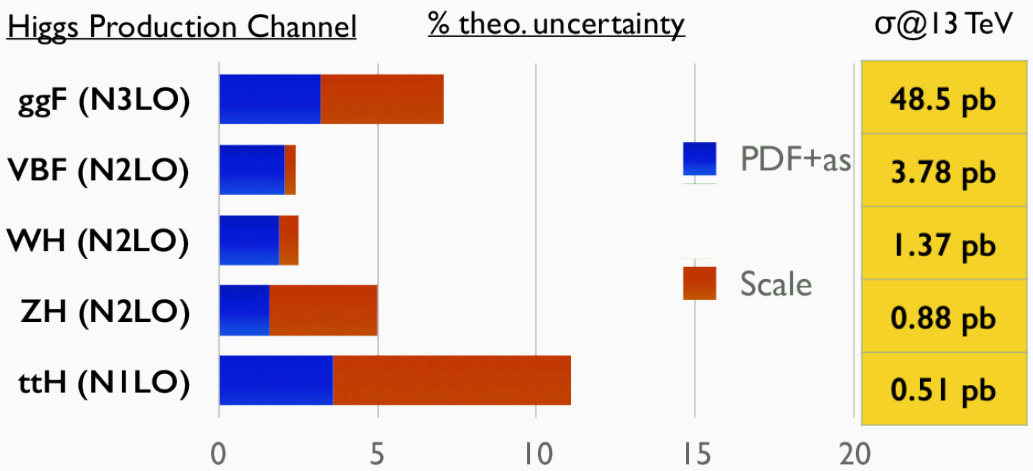




PDF uncertainties

PDF determination requires a sensible estimate of the **uncertainty**, and not only the central value, so not a well researched topic in ML.

CERN Yellow Report 4 (2016)



PDF uncertainties are a **limiting** factor in the accuracy of theoretical predictions for several processes at LHC.

$\Rightarrow$ Need of **precise** PDF determination and **uncertainty** estimate.

Why ML in PDFs determination?

- **PDFs** are **essential** for a **realistic computation** of hadronic particle physics **observable**, σ, thanks to the factorization theorem, *e.g.* in pp collider:

$$\underbrace{\sigma_X(s, M_X^2)}_{Y} = \sum_{a,b} \int_{x_{\min}}^{1} dx_1 dx_2 \; \underbrace{\hat{\sigma}_{a,b}(x_1, x_2, s, M_X^2)}_{X} \; f_a(x_1, M_X^2) f_b(x_2, M_X^2),$$

  where the elementary **hard cross-section** $\hat{\sigma}$ is convoluted with f the **PDF**.

- $f_i(x_1, M_X^2)$ is the PDF of parton i carrying a fraction of momentum x at scale M $\Rightarrow$ **needs to be learned from data.**

Why ML in PDFs determination?

- **PDFs** are **essential** for a **realistic computation** of hadronic particle physics **observable**, σ, thanks to the factorization theorem, *e.g.* in pp collider:

$$\underbrace{\sigma_X(s, M_X^2)}_{Y} = \sum_{a,b} \int_{x_{\min}}^{1} dx_1 dx_2 \, \underbrace{\hat{\sigma}_{a,b}(x_1, x_2, s, M_X^2)}_{X} \, f_a(x_1, M_X^2) f_b(x_2, M_X^2),$$

  where the elementary **hard cross-section** $\hat{\sigma}$ is convoluted with f the **PDF**.

- $f_i(x_1, M_X^2)$ is the PDF of parton i carrying a fraction of momentum x at scale M $\Rightarrow$ **needs to be learned from data.**

- Constraints come in the form of convolutions:

$$X \otimes f \to Y$$

- Experimental data points is $\sim 5000 \to$ not a big data problem
- Data from several process and experiments over the past decades $\Rightarrow$ deal with **data inconsistencies**

ML and PDF determination

The NNPDF methodology

The NNPDF (Neural Networks PDF) implements the Monte Carlo approach to the determination of a global PDF fit. We propose to:

1. **reduce** all sources of **theoretical bias**:
   - no fixed functional form
   - possibility to reproduce non-Gaussian behavior

   $\Rightarrow$ use Neural Networks instead of polynomials

2. provide a sensible estimate of the **uncertainty**:
   - uncertainties from input experimental data
   - minimization inefficiencies and degenerate minima
   - theoretical uncertainties

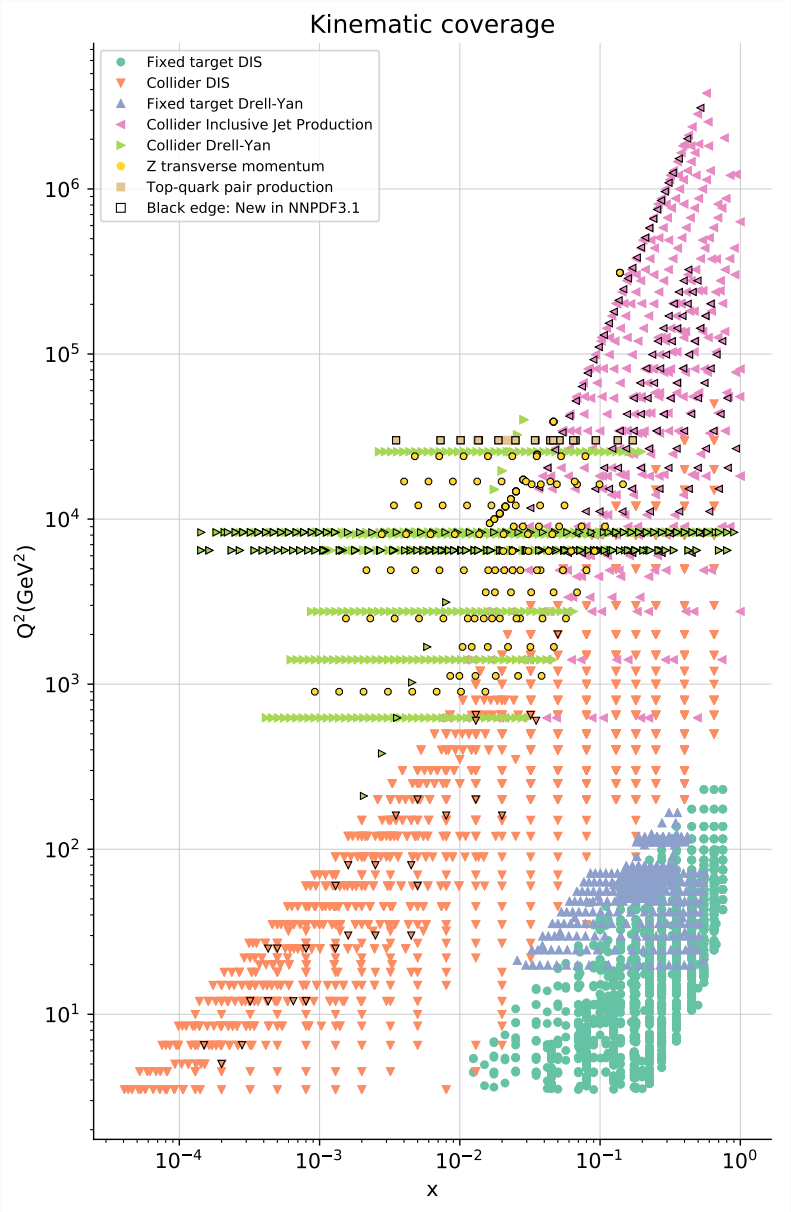
   $\Rightarrow$ use MC artificial replicas from data, training with a GA minimizer

3. Test the setup through **closure tests**

Experimental data

The total number of data points for the default PDF determination is

- 4175 at LO, 4295 at NLO and 4285 at NNLO.
- 7 physical processes from 14 experiments over ~30 years (deal with data inconsistencies)
- few data points at high and low x (deal with extrapolation)
- range of 5 and 7 orders of magnitude per PDF evaluation arguments (x, Q^2)



DGLAP evolution

Can we reduce the PDF input size? **Yes**, thanks to DGLAP:

$$f_i(x_\alpha, Q^2) = \Gamma(Q, Q_0)_{ij\alpha\beta} f_j(x_\beta, Q_0^2)$$

We remove the Q^2 dependence from PDF determination thanks to the DGLAP evolution operator Γ.

$$f(x, Q^2) \to f(x, Q_0^2) := f(x)$$

- Precompute the DGLAP operator for all data points
- Apply the operator to the partonic cross section
- Store the results and perform fast convolutions

In NNPDF theoretical predictions are stored in **APFELgrid** tables:

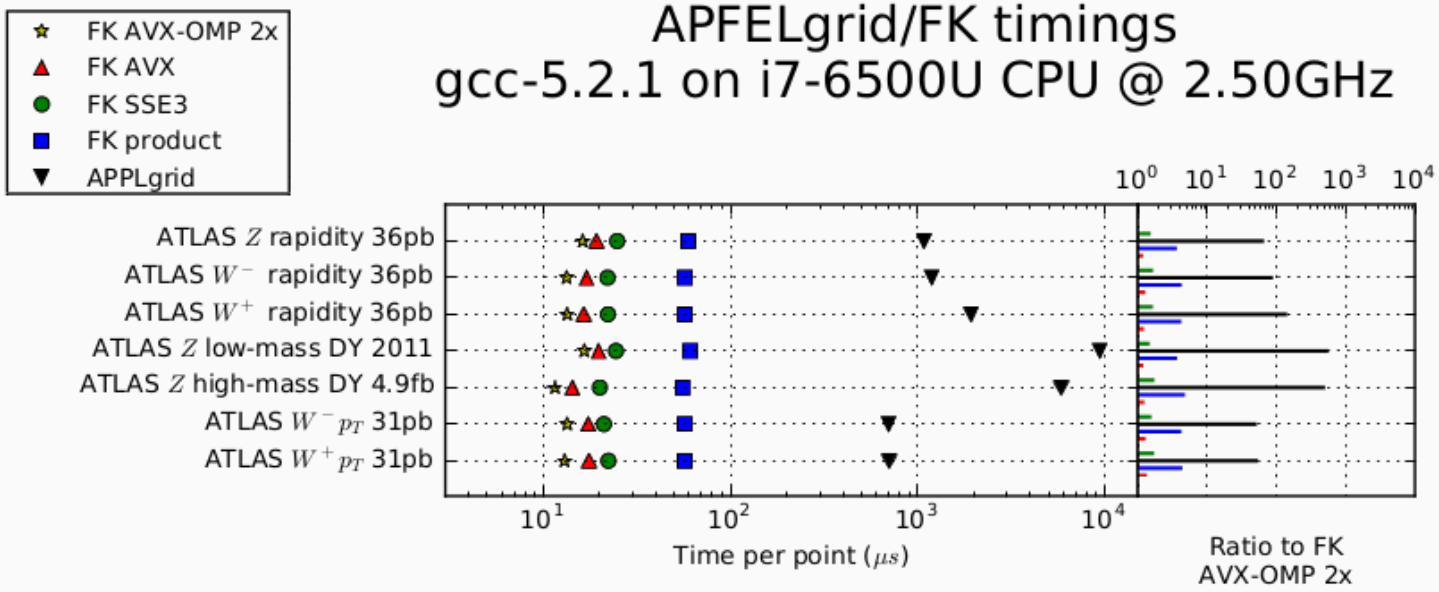
$$\sigma = \sum_{i,j}^{n_f} \sum_{\alpha,\beta}^{n_x} W_{ij\alpha\beta} f_i(x_\alpha, Q_0^2) f_j(x_\beta, Q_0^2)$$

Fast theory computation

APFELgrid (Bertone et al., arXiv:1605.02070) converts interpolated weight tables provided by APPLgrid in an efficient format for PDF fitting, *e.g.*

$$\sigma = \sum_{i,j}^{n_f} \sum_{\alpha,\beta}^{n_x} W_{ij\alpha\beta} f_i(x_\alpha, Q_0^2) f_j(x_\beta, Q_0^2)$$

where grids are pre-convoluted with PDF evolution kernels from **APFEL**.
(Bertone et al., arXiv:1310.1394)



Public code: https://github.com/nhartland/APFELgrid

Defining the ML problem

In comparison to a typical ML problem, a PDF fit

- requires a statistically sound uncertainty estimate
- is a regression problem but complex dependence on PDFs
- must satisfy physical constrains:
  - $f(x) \to 0$ for $x \to 1$ (continuity)
  - sum rules:

$$\sum_i^{n_f} \int_0^1 dx\, x f_i(x) = 1, \qquad \int_0^1 dx\, (u(x) - \bar{u}(x)) = 2$$

$$\int_0^1 dx\, (d(x) - \bar{d}(x)) = 1, \quad \int dx\, (q(x) - \bar{q}(x)) = 0,\ q = s, b, t$$

PDF parametrizations

- Early models:

$$f_i(x) = A \cdot x^{\alpha}(1-x)^{\beta}$$

  - parameters are chosen based on Hessian minimization approach
  - Can a simple model provide a reliable uncertainty estimate?
  - Can it deal with data inconsistencies?

PDF parametrizations

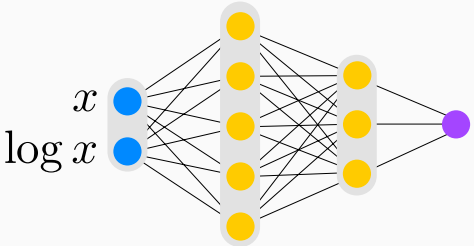
- Early models:

$$f_i(x) = A \cdot x^{\alpha}(1-x)^{\beta}$$

  - parameters are chosen based on Hessian minimization approach
  - Can a simple model provide a reliable uncertainty estimate?
  - Can it deal with data inconsistencies?

- NNPDF approach:

$$f_i(x, Q_0) = A \cdot x^{\alpha}(1-x)^{\beta} NN(x)$$



  - fully connected MLP (2-5-3-1)
  - two sigmoid hidden layers and linear output layer
  - x8 independent PDFs $\Rightarrow$ 296 free parameters

Cost function

- We minimize the cost function:

$$\chi^2 = \sum_{ij} (D_i - O_i) \sigma_{i,j}^{-1} (D_j - O_j)$$

  - D_i is the experimental measurement for point i
  - O_i the theoretical prediction for point i $(= \bar{\sigma} \otimes f)$
  - σ_{ij} is the covariance matrix between points i and j with corrections for normalization uncertainties
  - supplemented by additional penalty terms for positivity observables

Propagating experimental uncertainties

Generate artificial **Monte Carlo** data replicas from experimental data.

We perform N_{rep} O(1000) fits, sampling pseudodata replicas:

$$D_i^{(r)} \to D_i^{(r)} + \mathrm{chol}(\Sigma)_{i,j}\mathcal{N}(0,1), \quad i,j = 1..N_{\mathrm{dat}},\, r = 1...N_{\mathrm{rep}}$$

We obtain N_{rep} PDF replicas. No assumptions at all about the Gaussianity of the errors.

Propagating experimental uncertainties

Generate artificial **Monte Carlo** data replicas from experimental data.

We perform N_{rep} O(1000) fits, sampling pseudodata replicas:

$$D_i^{(r)} \to D_i^{(r)} + \mathrm{chol}(\Sigma)_{i,j}\mathcal{N}(0,1), \quad i,j = 1..N_{\mathrm{dat}},\, r = 1...N_{\mathrm{rep}}$$

We obtain N_{rep} PDF replicas. No assumptions at all about the Gaussianity of the errors.

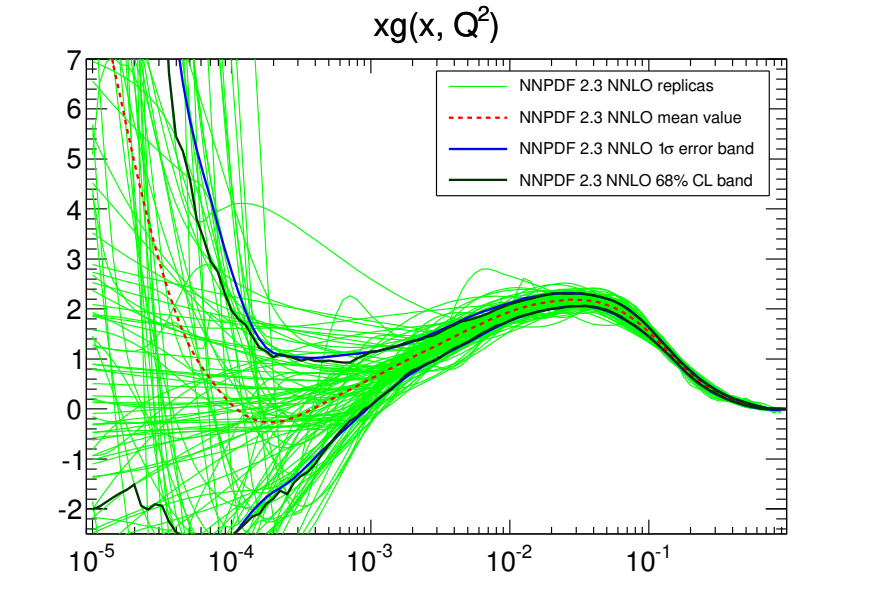
We perform compression techniques for PDF delivery:

- CMC-PDFs: compression algorithm for MC PDFs.
- mc2hessian: MC to hessian conversion tool for PDFs.
- SMPDF: Specialized Minimal PDFs.

PDF releases reduce 1000 replicas to 100.

PDF fit example

The procedure delivers a **Monte Carlo** representation of results:



The central value of observables based on PDFs are obtained with:

$$\langle \mathcal{O}[f] \rangle = \frac{1}{N_{\rm rep}} \sum_{k=1}^{N_{\rm rep}} \mathcal{O}[f_k]$$

Optimization algorithm

The current approach is genetic optimization, based on nodal mutation probabilities and more recently the covariance matrix evolution strategy

$$w \to w + \eta \frac{r_\delta}{N_{\text{ite}}^{r_{\text{ite}}}}, \quad \eta = 15,\, r_\delta \sim U(-1,1),\, r_{\text{ite}} \sim U(1,0)$$

At each iteration, generate 80 mutants and select best mutant.

Advantages

- Simple to implement and understand.
- Good dealing with complex analytic behavior.
- Doesn't require evaluating the gradient.

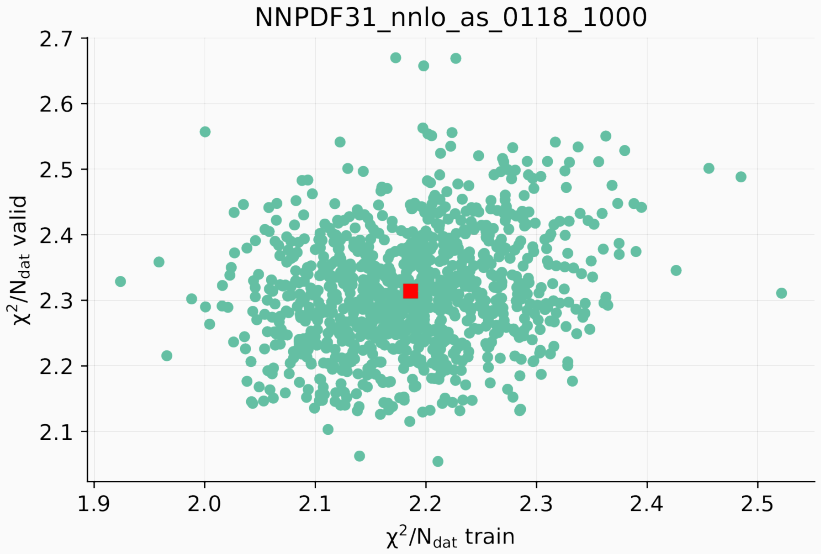
Disadvantages

- May not be close to a global minimum.
- Requires many functions evaluations.
- Needs tuning.

Stopping

We have cross-validation implemented:

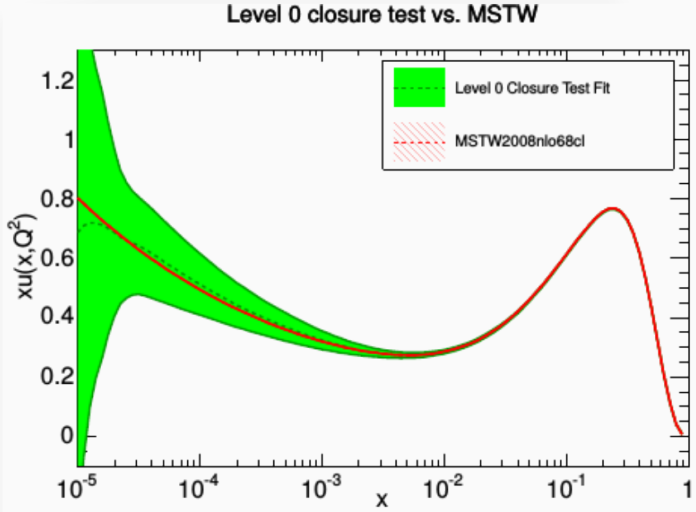
- We split data in a training and validation set.
- Training fraction is 50%, different for each replica.
- We perform the GA on the training set for a fixed number of iterations O(30000).
- Stop at the minimum of the validation set, storing the parameters from the replica at that iteration.



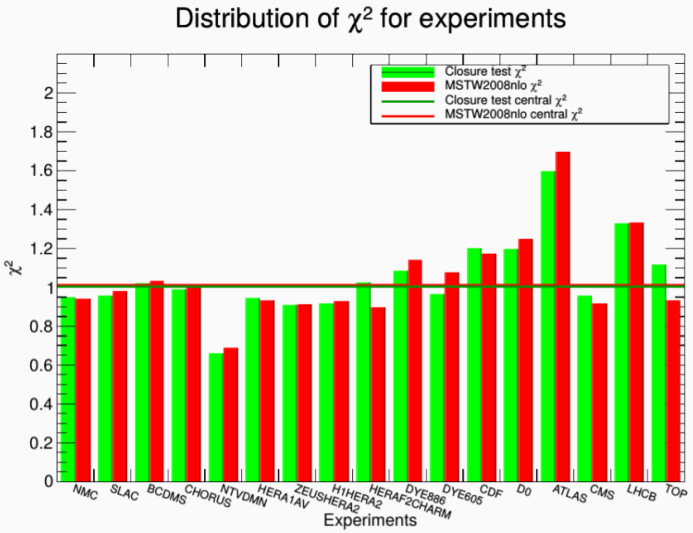
Validation with closure test

Closure tests

- Assume that the underlying PDF is known, generate data, fluctuations around the prediction of the true PDF.
- Perform a fit and compare to underlying PDF.
- Check that the results are consistent.



(a) Level 0 fit



(b) Level 2 results

Level 0: Fit predictions of the true PDF without fluctuations. $\chi^2/N_{\text{dat}} \to 0$.

Level 2: Generate pseudodata replicas on top of replicas. $\chi^2/N_{\text{dat}} \to 1$.

Summary

Summary

We have covered the following topics:

- The hyperparameter tune
- the cross-validation techniques
- ML frameworks
- The PDF case study