Analytical derivates of neural networks: making a fit faster

Valerio Bertone

INFN and Università di Pavia

(in collaboration with Rabah Khalek)

Istituto Nazionale di Fisica Nucleare

UNIVERSITÀ DI PAVIA

European Research Council

Established by the European Commission

xFitter External Meeting 2019

March 19, 2019, Minsk

3DSPIN MAPPING THE PROTON IN 3D

Defining the problem

- A fit usually entails the **minimisation** of the χ^2:

$$\chi^2 = \left(\frac{\hat{F} - F}{\sigma} \right)^2$$

Defining the problem

- A fit usually entails the **minimisation** of the χ^2:

Theoretical prediction

$$\chi^2 = \left(\frac{\boxed{\hat{F}} - \boxed{F}}{\sigma}\right)^2$$

Experimental data

Defining the problem

- A fit usually entails the **minimisation** of the χ^2:

Theoretical prediction

$$\chi^2 = \left(\frac{\boxed{\hat{F}} - \boxed{F}}{\sigma}\right)^2$$

Experimental data

- The theoretical prediction is usually computed as a **convolution**:

$$\hat{F} \equiv \sum_i C_i \otimes N_i = \mathbf{C} \otimes \mathbf{N}$$

Defining the problem

- A fit usually entails the **minimisation** of the χ^2:

Theoretical prediction

Experimental data

$$\chi^2 = \left(\frac{\boxed{\hat{F}} - \boxed{F}}{\sigma} \right)^2$$

- The theoretical prediction is usually computed as a **convolution**:

Hard cross sections
times evolution
(APFELgrid)

Initial-scale distributions

$$\hat{F} \equiv \sum_i \boxed{C_i} \otimes \boxed{N_i} = \mathbf{C} \boxed{\otimes} \mathbf{N}$$

Integral

Defining the problem

- A fit usually entails the **minimisation** of the χ^2:

Theoretical prediction

$$\chi^2 = \left(\frac{\boxed{\hat{F}} - \boxed{F}}{\sigma}\right)^2$$

Experimental data

- The theoretical prediction is usually computed as a **convolution**:

Hard cross sections
times evolution
(APFELgrid)

Initial-scale distributions

$$\hat{F} \equiv \sum_i \boxed{C_i} \otimes \boxed{N_i} = \mathbf{C} \boxed{\otimes} \mathbf{N}$$

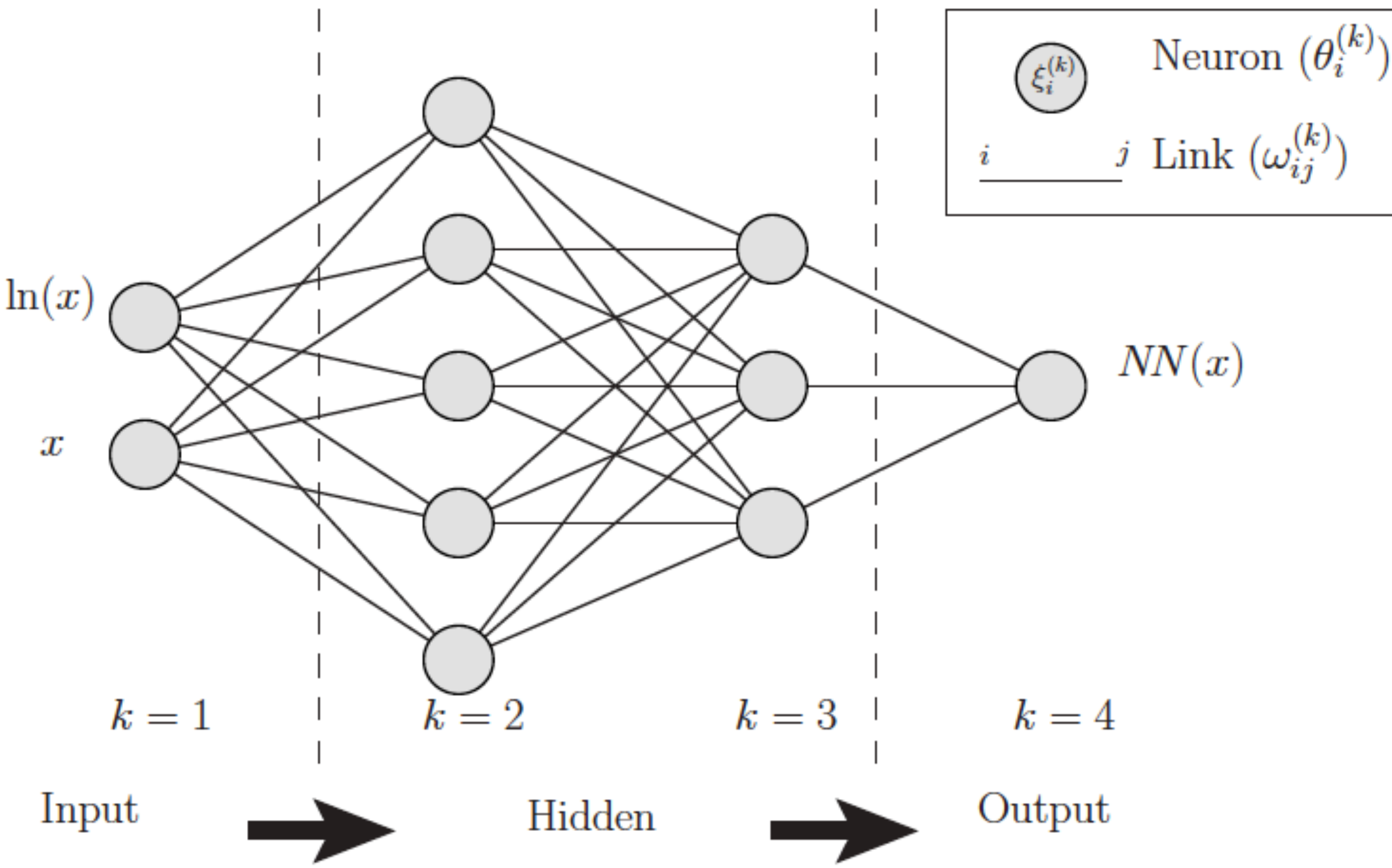
Integral

- Therefore the theoretical prediction is a **functional** of the initial-scale distributions and so is the χ^2:

$$\hat{F} \equiv \hat{F}[\{N_i\}] \quad \text{and} \quad \chi^2 \equiv \chi^2[\{N_i\}]$$

Defining the problem

- A feed-forward neural network (NN) is parameterised in terms of **links** and **biases**, *e.g.*:



$$\xi_i^{(j)} = \sigma\left(\sum_k^{\text{(j-1)th layer}} \xi_k^{(j-1)} \omega_{ki}^{(j)} - \theta_i^{(j)}\right)$$

Activation function associated to each neuron

- Effectively, a NN is nothing but a **parametric function**:

$$\begin{aligned} N_i \equiv N_i(x; \{\omega_{ij}^{(\ell)}, \theta_i^{(\ell)}\}) &= \sigma_L\left(\sum_{j^{(1)}=1}^{N_{L-1}} \omega_{ij^{(1)}}^{(L)} y_{j^{(1)}}^{(L-1)} + \theta_i^{(L)}\right) \\ &= \sigma_L\left(\sum_{j^{(1)}=1}^{N_{L-1}} \omega_{ij^{(1)}}^{(L)} \sigma_{L-1}\left(\sum_{j^{(2)}=1}^{N_{L-2}} \omega_{j^{(1)}j^{(2)}}^{(L)} y_{j^{(2)}}^{(L-2)} + \theta_{j^{(1)}}^{(L-1)}\right) + \theta_i^{(L)}\right) \\ &= \ldots \end{aligned}$$

Defining the problem

- In order to minimise the χ^2 **optimally**, we want to be able to compute the following derivatives:

$$\frac{\partial \chi^2}{\partial \omega_{ij}^{(\ell)}} = 2\left(\frac{\mathbf{C}\otimes\mathbf{N} - F}{\sigma^2}\right)\mathbf{C}\otimes\frac{\partial \mathbf{N}}{\partial \omega_{ij}^{(\ell)}}$$

$$\frac{\partial \chi^2}{\partial \theta_{i}^{(\ell)}} = 2\left(\frac{\mathbf{C}\otimes\mathbf{N} - F}{\sigma^2}\right)\mathbf{C}\otimes\frac{\partial \mathbf{N}}{\partial \theta_{i}^{(\ell)}}$$

- This boils down to computing the **derivative of the NN** w.r.t. to its free parameters.
- We can surely do it **numerically** (incremental ratio).
- Can we also do it **analytically** for a generic architecture?
  - better **numerical stability**,
  - **faster**.

Deriving a Neural Network

- Yes, we can derive a feed-forward NN by using the **chain rule**:

$$\frac{\partial N_k}{\partial \theta_i^{(\ell)}} = \Sigma_{ki}^{(\ell)} z_i^{(\ell)} \qquad \text{and} \qquad \frac{\partial N_k}{\partial \omega_{ij}^{(\ell)}} = \Sigma_{ki}^{(\ell)} z_i^{(\ell)} y_j^{(\ell-1)}$$

with:

$$x_i^{(\ell)} = \sum_{j=1}^{N_{\ell-1}} \omega_{ij}^{(\ell)} y_j^{(\ell-1)} + \theta_i^{(\ell)} \,,$$

$$\boldsymbol{\Sigma}^{(\ell)} = \prod_{\alpha=L}^{\ell+1} \mathbf{S}^{(\alpha)}$$

$$y_i^{(\ell)} = \sigma_\ell\left(x_i^{(\ell)}\right) \,,$$

$$z_i^{(\ell)} \omega_{ij}^{(\ell)} = S_{ij}^{(\ell)} \left(= \frac{\partial y_i^{(\ell)}}{\partial y_j^{(\ell-1)}} \right)$$

$$z_i^{(\ell)} = \sigma'_\ell\left(x_i^{(\ell)}\right) \,,$$

Tests against numerical derivatives show that it works beautifully!

Exploiting analytic derivatives

- Use **APFEL/APFELgrid** as it give access to the initial-scale PDFs:

$$F(Q) = \underbrace{C(Q) \otimes \Gamma(Q, \mu_0)}_{\text{FK table}} \otimes f(\mu_0)$$

- Use **ceres-solver** to assess the impact of different derivation strategies:
  - **analytic** derivatives,
  - **numerical** derivatives (incremental ratio: forward/backward/central),
  - **automatic** derivatives:

Dual Numbers & Jets

Dual numbers are an extension of the real numbers analogous to complex numbers: whereas complex numbers augment the reals by introducing an imaginary unit ι such that $\iota^2 = -1$, dual numbers introduce an *infinitesimal* unit ϵ such that $\epsilon^2 = 0$. A dual number $a + v\epsilon$ has two components, the *real* component a and the *infinitesimal* component v.

http://ceres-solver.org/automatic_derivatives.html#chapter-automatic-derivatives

Dual numbers and Jets

For example, consider the function

$$f(x) = x^2,$$

Then,

$$\begin{aligned} f(10+\epsilon) &= (10+\epsilon)^2 \\ &= 100 + 20\epsilon + \epsilon^2 \\ &= 100 + 20\epsilon \end{aligned}$$

Observe that the coefficient of ϵ is $Df(10) = 20$. Indeed this generalizes to functions which are not

A Jet is a n-dimensional dual number, where we augment the real numbers with n infinitesimal units $\epsilon_i,\ i = 1, \ldots, n$ with the property that $\forall i, j\ : \epsilon_i \epsilon_j = 0$. Then a Jet consists of a *real* part a and a n-dimensional *infinitesimal* part $\mathbf{v}$, i.e.,

$$x = a + \sum_j v_j \epsilon_j$$

The summation notation gets tedious, so we will also just write

$$x = a + \mathbf{v}.$$

where the ϵ_i's are implict. Then, using the same Taylor series expansion used above, we can see that:

$$f(a + \mathbf{v}) = f(a) + Df(a)\mathbf{v}.$$

Automatic derivation

- Automatic derivatives are available **through ceres** if one **templates** the function to be derived in such a way to **evaluate** it, not just on real numbers, but also **on dual numbers/jets**:

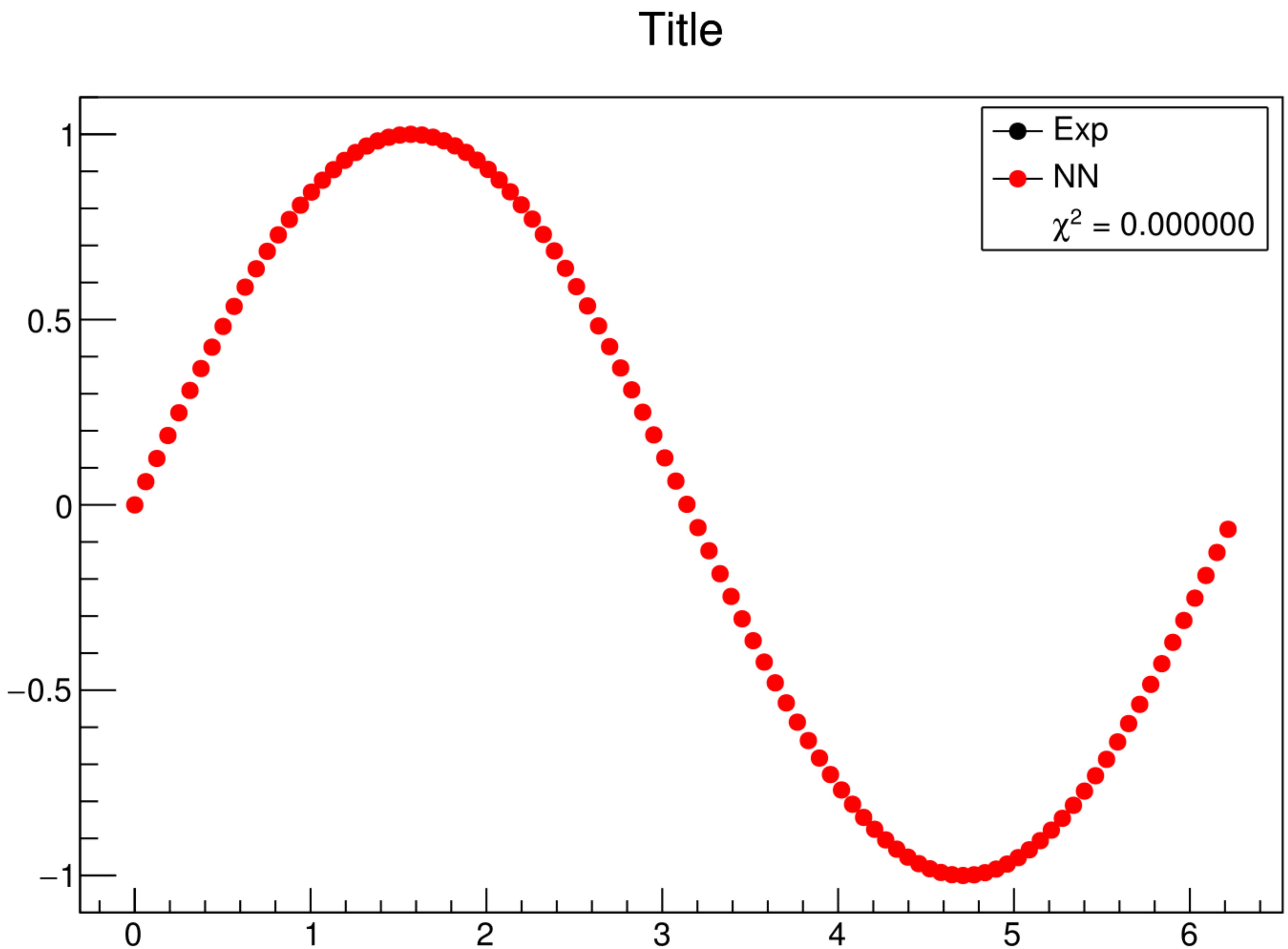
```
template<class T>
class FeedForwardNN
{
 public:

  // Constructors
  FeedForwardNN(std::vector<int> const &Arch,
                int const &RandomSeed = 0,
                std::function<T(T const &)> const &ActFun = Sigmoid<T>,
                std::function<T(T const &)> const &dActFun = dSigmoid<T>,
                bool const &LinearOutput = true);
```

- During the fit ceres takes care of computing the derivatives w.r.t. the free parameters by **evaluating the (jet-valued) function on a jet**.

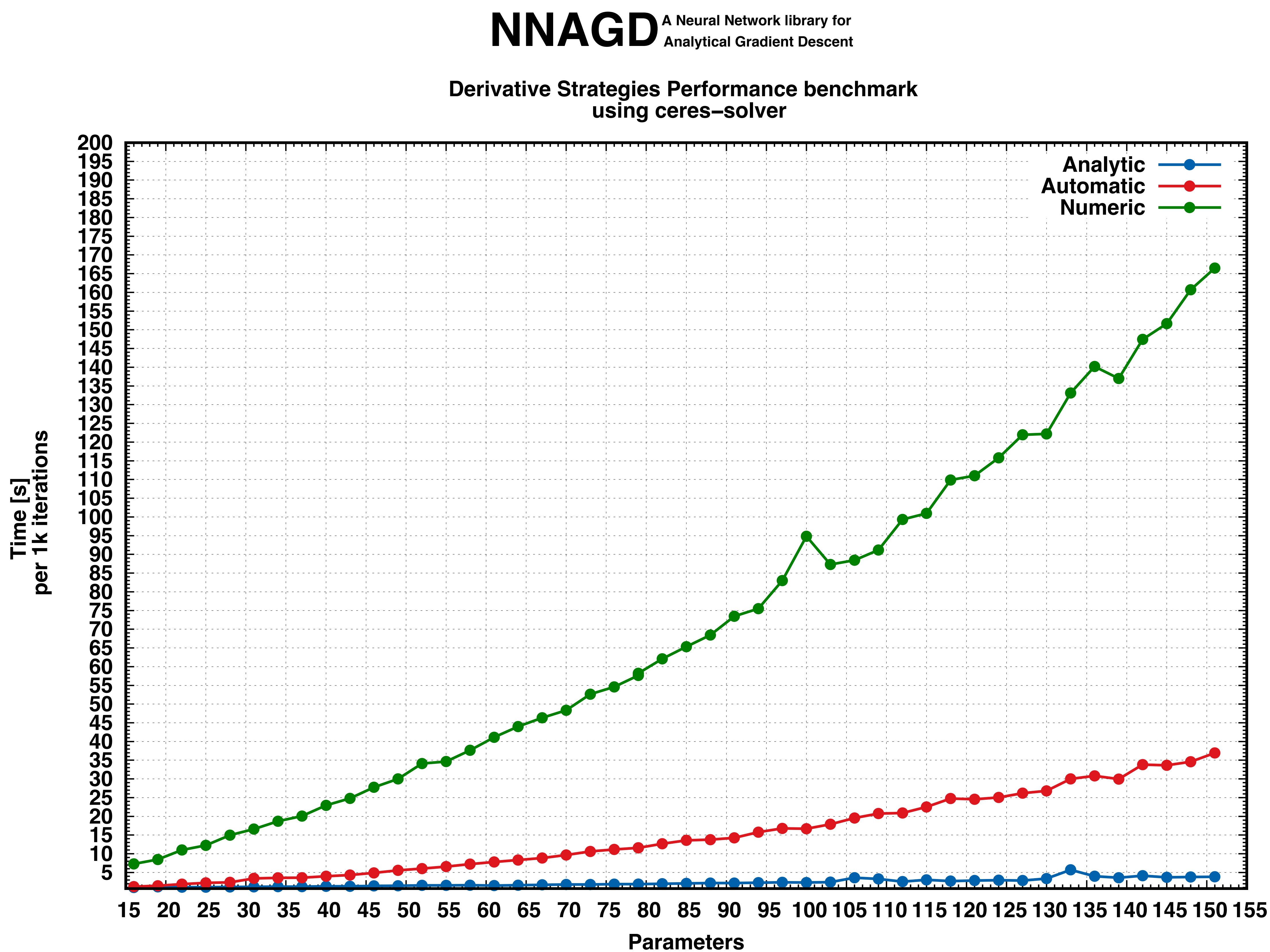
Performance

- Fit 100 points of a **sine function** using a NN with one single hidden layer with an increasing number of nodes:

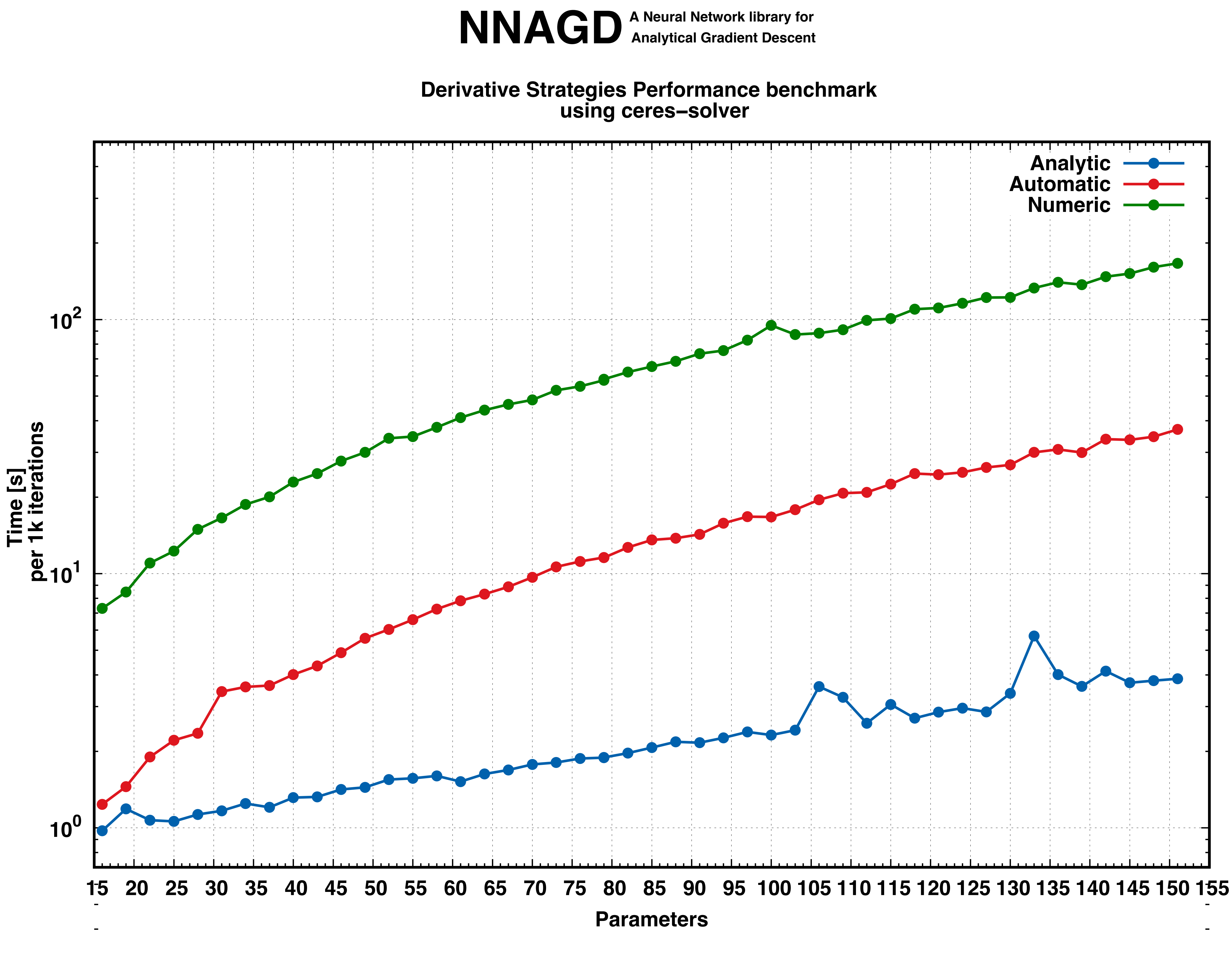


- Let the code run for **1k iterations** ($\chi^2 \simeq 10^{-9}$ - 10^{-12}).

Performance

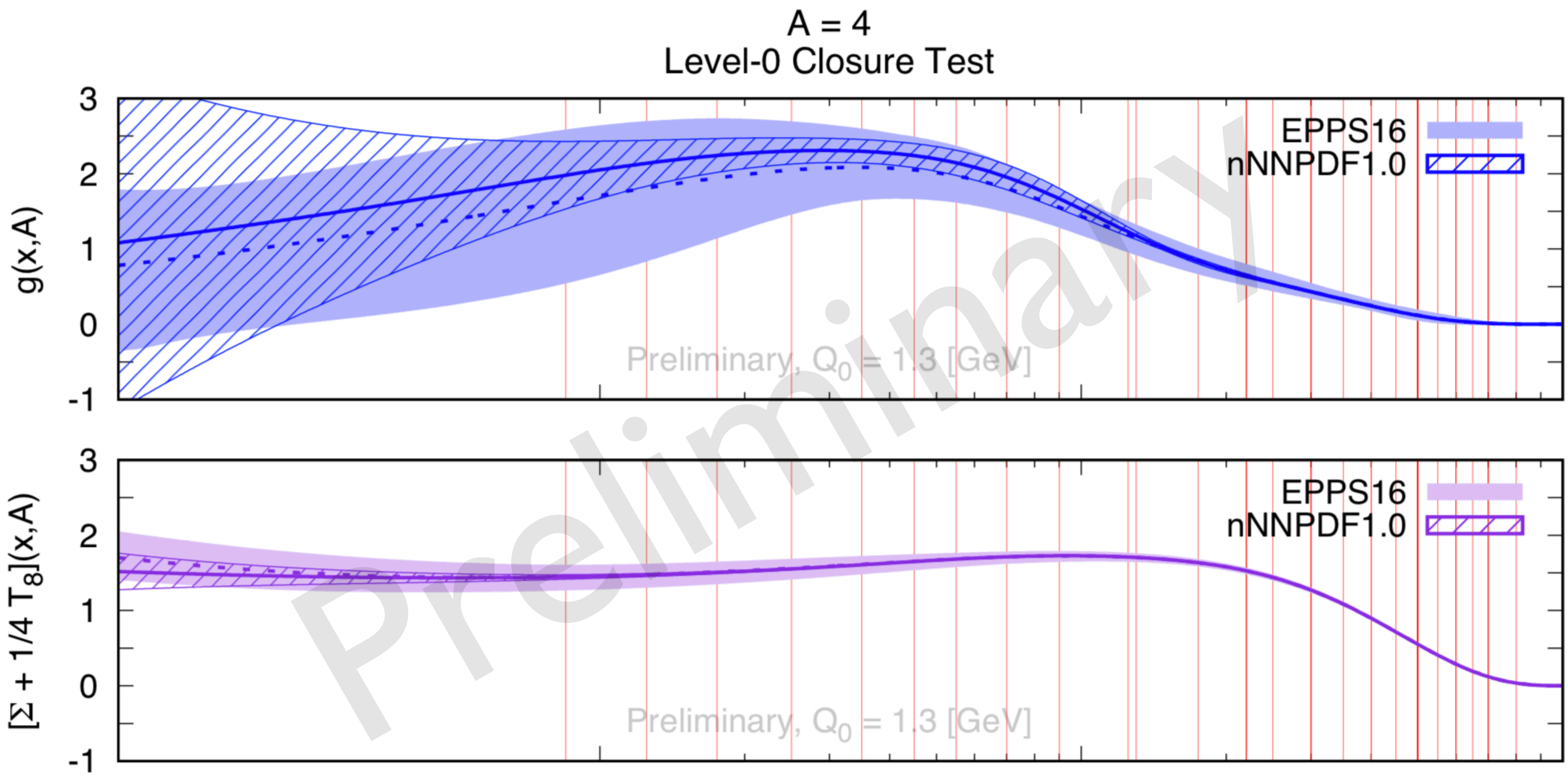


Performance



Preliminary closure tests

- Use **analytic derivatives** of a feed-forward NN to extract nuclear PDFs from **DIS pseudo-data** (closure test) using **APFELgrid** and **ceres**:



Summary

- Feed-forward NNs can be **derived analytically** by using the chain rule,
- this allows one to compute the gradient of the χ^2 analytically:
  - provided that one uses an **APFELgrid-like computation**.
- The analytic knowledge of the gradient of the χ^2 makes a fit much faster.
- We have used **ceres** to assess the performance of analytic, numeric, and automatic differentiation.
- Analytic derivatives seem to perform substantially better.