DAML Sem 2 Week 6, CP10: Determing the momentum resolution for muon tracks at LHCb with NN Regression

Christos Leonidopoulos[†]

University of Edinburgh

February 22, 2024

1 Introduction

In the last Semester we looked at *classification* problems with Neural Networks (NN), where the goal was to predict the category (or: label) of a given measurement (*e.g.* type of weather or particle identification). We will now switch our attention to a different type of Machine Learning (ML) problem: *regression*, where the goal is to make a prediction about a continuous variable[*], in a way very similar to a fitting problem, where we extrapolate (or interpolate) the fitted function into a region of interest. The major difference in ML regression is that the problem is typically multi-dimensional, and we do not attempt to solve it with an analytical function.

We will be using simulated data from the LHCb experiment [1] at CERN. We will explore how to use NN for regression, and evaluate the performance of different architectures.

2 Muons from χ_{c_1} decays with LHCb

The $\chi_{c_1} \to J/\psi(\mu^+\mu^-)\,\mu^+\mu^-$ decays are observed in the collision data collected with the LHCb experiment, and used to study the resonance parameters of the χ_{c_1} mesons [2].

Muon tracks from χ_{c_1} decays are fitted, and their momentum measured typically at the few per mille level. The LHCb detector has several features that makes the tracking

[†]`Christos.Leonidopoulos@ed.ac.uk`

[*]It is not a strict requirement that the variable is continuous; for example, it can be an integer over a large enough range. However, for the needs of the regression algorithm we always treat the variable as continuous, even if it isn't.

performance (*i.e.* the tracking efficiency and the momentum resolution of these muon tracks) very non-homogeneous, *e.g.*

- The silicon tracker detector has a limited acceptance, resulting into a coverage gap at high η[†].
- The VELO tracking detector opens and closes in two halves around $x = 0$.
- The detector *material budget* (*i.e.* equivalent to the number of radiation lengths X_0 a particle encounters as it transverses the detector) is heterogeneous in η and ϕ.
- The starting point of the beampipe has an aperture of 25 mrad, giving a spike in the material budget.

The momentum resolution for muons from χ_{c_1} decays can be seen in Fig. 1.

In tracking systems, the expected momentum resolution typically follows a relationship of the form

$$\frac{\sigma_p}{p} = A + B \times p \tag{1}$$

where σ_p/p is the relative momentum resolution, expressed as the sum of a constant term (A) which corresponds to multiple-scattering and depends on the radiation length the particles encounter while traversing the detector, and a linear term ($B \times p$) which corresponds to the curvature measurement errors of the (charged) particles as they travel inside the magnetic field. A more detailed expression for the dependence of the momentum resolution on detector parameters can be seen in Fig. 2.

Note that "momentum" here refers to the component of the 3D vector which is orthogonal to the magnetic field. The LHCb detector is in the forward region, so the magnetic field is set up in a way to measure the z-component of the momentum[‡]. Since $p_T \ll p_z$, we could be using either p_z or p for the purposes of the data analysis: the difference between the two variables is negligible.

The major goal of this checkpoint will be to try to predict (*i.e.* model) the σ_p/p dependence shown in Fig. 1, given a number of features contained in the dataset, so it can be potentially used as input to a simplified simulation. This is because the "full" (GEANT-type) simulation and the track-fitting components of a study are generally very CPU-intensive, and the cost can be prohibitive when generating a large number of events. Applications of this approach could be *e.g.* studies of different detector designs to be used for evaluating the impact of the momentum resolution on some measurement, to be used in deciding if a more granular detector (offering an improved momentum resolution) is worth the extra cost.

[†]LHCb uses a right-handed coordinate system with its origin at the nominal interaction point (IP) in the centre of the detector and the z-axis along the beam pipe. The x-axis points form the IP to the centre of the LHC ring, and the y-axis points upwards. Cylindrical coordinates (r, ϕ) are used in the transverse plane, ϕ being the azimuthal angle around the z-axis. The pseudorapidity is defined in terms of the polar angle θ as $\eta \equiv -\log\tan(\theta/2)$.

[‡]This is different than in the ATLAS and CMS experiments, which have their magnetic fields set up in a way to measure the transverse momentum, p_T.

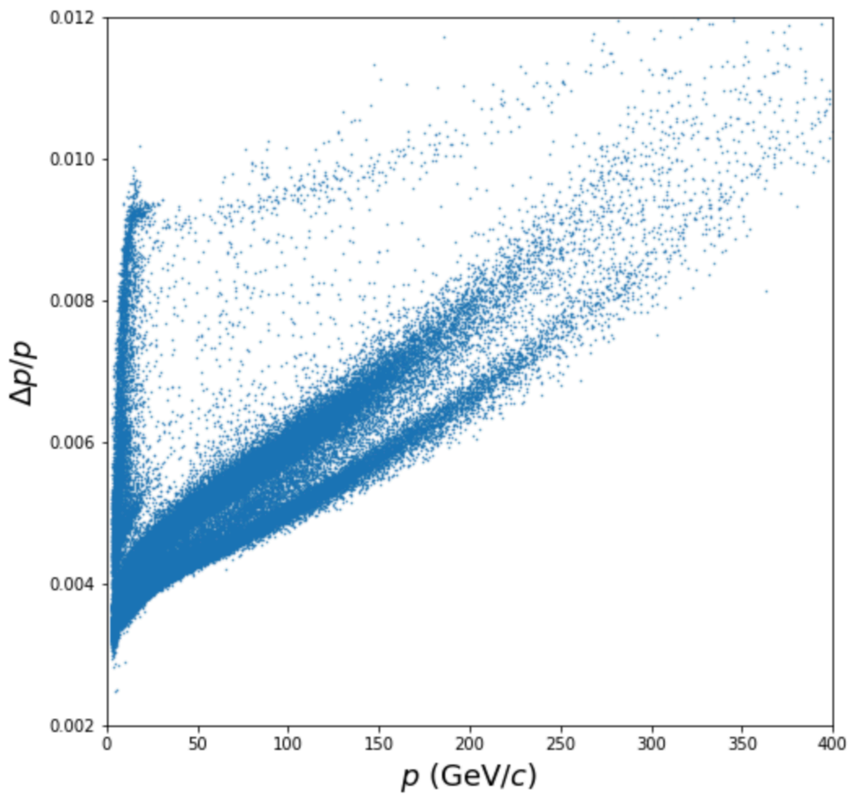


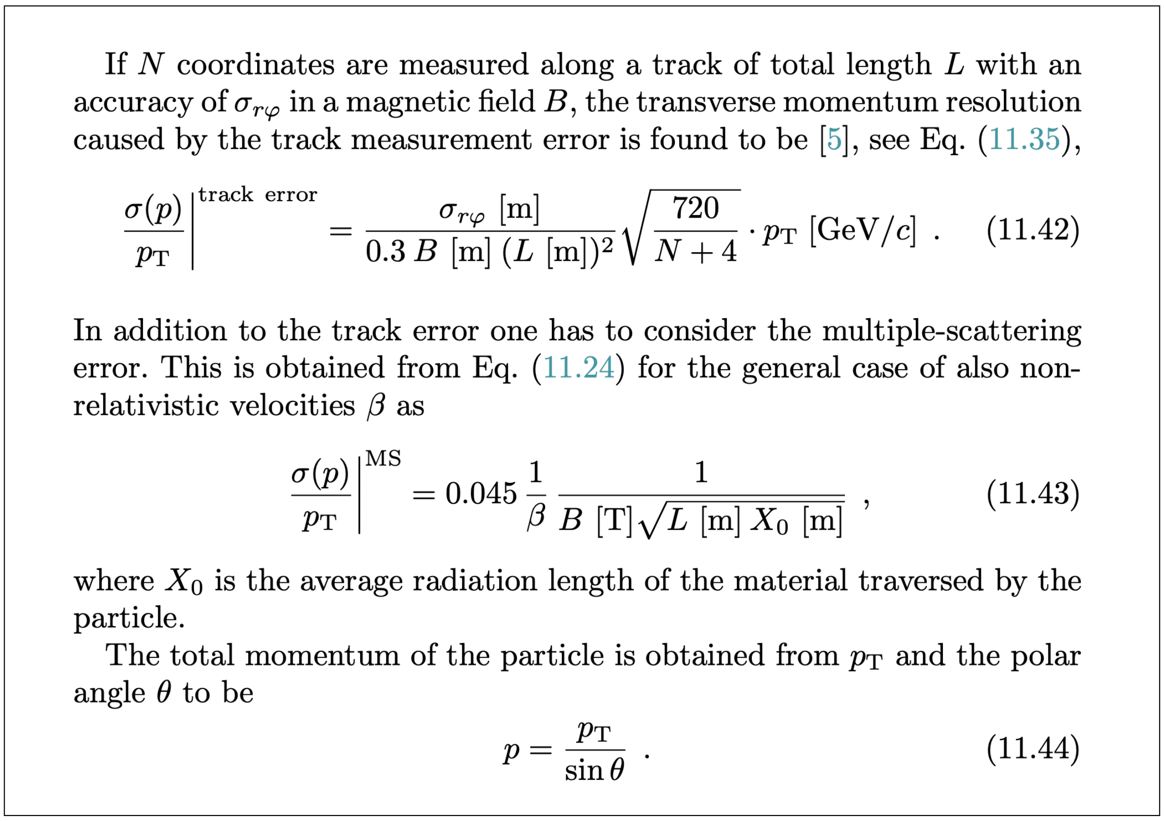
Figure 1: The fitted muon track momentum resolution ($\Delta p/p$) as a function of the muon momentum, p. There is a clear structure in the scatter plot with 3-4 distinct bands, with each one of them corresponding to tracks traversing different parts of the detector. Each of these bands appears to be following a linear pattern. See text for details.

We will be working with a simulated dataset containing kinematic information from the fitted tracks of the muons in the $\chi_{c_1} \to J/\psi\, \mu^+\mu^-$ decay.

3 The muon dataset

A CSV file with the kinematic information for about 200k muons from simulated χ_{c_1} decays can be found at `https://cernbox.cern.ch/s/WikS3ALBku9CkZH`. There is a separate entry in the table for each muon track.

The variables contained in the CSV file are summarised in Table 1. Some of the information stored in these variables is redundant, *e.g.* there are more than one ways of calculating p_z.



If N coordinates are measured along a track of total length L with an accuracy of $\sigma_{r\varphi}$ in a magnetic field B, the transverse momentum resolution caused by the track measurement error is found to be [5], see Eq. (11.35),

$$\left.\frac{\sigma(p)}{p_\mathrm{T}}\right|^{\text{track error}} = \frac{\sigma_{r\varphi}\ [\mathrm{m}]}{0.3\,B\ [\mathrm{m}]\ (L\ [\mathrm{m}])^2}\sqrt{\frac{720}{N+4}}\cdot p_\mathrm{T}\ [\mathrm{GeV}/c]\ . \qquad (11.42)$$

In addition to the track error one has to consider the multiple-scattering error. This is obtained from Eq. (11.24) for the general case of also non-relativistic velocities β as

$$\left.\frac{\sigma(p)}{p_\mathrm{T}}\right|^{\mathrm{MS}} = 0.045\,\frac{1}{\beta}\,\frac{1}{B\ [\mathrm{T}]\sqrt{L\ [\mathrm{m}]\,X_0\ [\mathrm{m}]}}\ , \qquad (11.43)$$

where X_0 is the average radiation length of the material traversed by the particle.

The total momentum of the particle is obtained from p_T and the polar angle θ to be

$$p = \frac{p_\mathrm{T}}{\sin\theta}\ . \qquad (11.44)$$

Figure 2: Contributions to track measurement uncertainty from measurement errors, proportional to the transverse momentum, p_T (Eq. 11.42) and the constant term from multiple scattering (Eq. 11.43). Credit: Ref. [4].

Variable	Description
`ep`	estimate of relative muon track fit momentum error (σ_p/p)
`eta`	muon track η
`p`	muon track momentum magnitude in GeV/c ($p \equiv \sqrt{p_x^2+p_y^2+p_z^2}$)
`phi`	muon track ϕ in rad
`pol`	magnet polarity ($+$ or $-$)
`pt`	muon track transverse momentum in GeV/c ($p_T \equiv \sqrt{p_x^2+p_y^2}$)
`qp`	muon charge $\times$ momentum in GeV/c
`tx`	ratio of x- to z-momentum components of muon track (p_x/p_z)
`ty`	ratio of y- to z-momentum components of muon track (p_y/p_z)
`zV`	z-component of χ_{c1} decay vertex in mm

Table 1: List of muon tracking variables contained in CSV file and brief description.

4 Data inspection and variable plotting

We begin with the usual and necessary sanity checks we carry out upon data unpacking. In order to make sure we understand the data that we are dealing with, we usually print out raw information for a handful of entries on the console, and plot distributions to get a feel of the problem at hand.

Exercise #1 (1 point):

Open the CSV file, and feed the data into a `pandas.core.frame.DataFrame` object. Do a sanity check of the dataset: Is the definition for all variables clear? Do all variables have the same number of entries? Are there any null values that need to be discarded?

Reproduce the scatter plot of Fig. 1. Make sure to zoom in the plot in a way that allows you to see clearly the different momentum resolution bands.

Hint #1: you may want to review the material taught in last semester (on Data Science Tools) for dealing with questions on data handling and variable plotting.

Exercise #2 (1 point):

Demonstrate that $p_T \ll p_z$ in two ways:

- Plot the 1D distributions for p and p_T in log scale.
- Use the "data-appending" method (`pandas.DataFrame.append`) to add the variables `pz` (*i.e.* $p_z \equiv \sqrt{p^2 - p_T^2}$) and `epz` (= `ep * p/pz`, *i.e.* effectively σ_{p_z}/p_z) in the `DataFrame` object. Create the σ_{p_z}/p_z *vs.* p_z scatter plot and demonstrate that it is qualitatively identical to Fig. 1.

Hint #2: It is good practice to make a copy of the `DataFrame` *object before appending data (in case something goes wrong).*
Hint #3: You may want to use method `pandas.DataFrame.iterrows` *in your data-appending operation.*

5 Regression with Neural Networks

We will now try to model the σ_p/p dependence by using regression employed with various Neural Network architectures. We will use as input features the following kinematic variables: `p`, `tx`, `ty`, `eta` and `phi`. `ep` is the *target* variable, *i.e.* the variable whose behaviour we are trying to predict.

We will start with a simple, one-layer NN, and gradually "upgrade" the implementation of the regression algorithm to more advanced architectures. We will evaluate each design with a common metric in order to identify the most performant architecture.

For the purposes of this checkpoint, we will be using the `keras` [5] library that allows the implementation of NN networks with a few lines of code. As a first step, we must prepare the data. We do this as usual, by creating a new dataframe containing only the features of interest[§], as was discussed in the ML checkpoints of the previous Semester.

Exercise #3 (2 points):

Define a simple single-layer Keras model in a function that uses the same number of nodes as the number of inputs.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

# num of inputs = 5 (p, tx, ty, eta, phi), num of outputs = 1 (ep)
num_inputs = 5; num_outputs = 1

# simple Keras model: use same # of nodes as # of inputs, single layer
num_nodes = num_inputs

def simple_model():
    # create model
    model = Sequential()
    # no activation required for the output, as this is a regression problem,
    # ie. we need a numerical prediction for any input
    model.add(Dense(num_nodes, input_dim=num_inputs, kernel_initializer='normal', \
    activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_outputs, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

Create a `Keras Regressor` using the simple model coded up above.

We are going to use a 10-fold cross validation, and take the average score of the `'r2'` method [6] to evaluate the performance of the regression implemented with this initial simple model.
Hint #4: You can safely ignore warnings that package `KerasRegressor` *is deprecated.*

```
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

N_epochs = ???
batchSize = ???

# must always set the random seed for reproducibility
Answer_to_all_questions = 42
np.random.seed(Answer_to_all_questions)
estimator = KerasRegressor(build_fn=simple_model, epochs=N_epochs, batch_size=batchSize)
kfold = KFold(n_splits=10, random_state=Answer_to_all_questions, shuffle=True)
# data and target are 5-column and 1-column arrays produced with pandas.DataFrame.values
results = cross_val_score(estimator, data, target, cv=kfold, scoring='r2')
print("Standardised: %.2f %s %.2f" % (results.mean(), u"\u00B1", results.std()))
```

Play with the number of epochs and the batch size. You can get a feel about how fast the estimator is converging if you switch to `verbose=1` in the `KerasRegressor` call. Set the number of epochs to 100 as a starting point. Notice (by eye) how quickly the loss function converges as you change the batch size from (say) 5 to (say) 1,000. After you

[§] Use method `pandas.DataFrame.values` (or `pandas.DataFrame.to_numpy`) to create separate `numpy` arrays with the input and target data to be fed into the NN.

Exercise #3 (*cont.*):

pick a pair of (epochs, batch-size) values you are comfortable with, you can keep them fixed for the rest of this exercise.

Hint #5: The options in the code above are simply suggestions. You are very welcome (in fact, encouraged) to try out different options [7] and/or discuss your ideas with the TAs during the workshop hours.

One of the NN peculiarities is that a regressor may get confused (and its performance affected) when the input features have very different numerical ranges. To remedy for this, we will be "standardising" the data by using a "pipeline" of transforms that adjust the input data to have the same range.

Exercise #4 (3 points):

(a) **(1 point)** In the code of the previous example, we will be replacing the estimator created via the `KerasRegressor` call by a new "pipelined" estimator using the `sklearn.pipeline.Pipeline` method.

```
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=simple_model, \
  epochs=N_epochs, batch_size=batchSize))
pipeline = Pipeline(estimators)
```

The rest of the code remains the same. Evaluate the performance of the new estimator with standardised inputs. How does it perform compared to the previous example?

(b) **(1 point)** We will now up our game and attempt to build a denser model, *i.e.* a model with an additional NN layer.

Create a new `denser_model` based on the `simple_model` function, and by adding a new layer. For the first layer, use the same number of nodes as the number of inputs. For the second layer, choose a number of nodes anywhere between 2 and the number of inputs.

Run the new regressor and evaluate again its performance. Should you use standardised inputs again? Only if the previous exercise has shown that this improves the performance!

(c) **(1 point)** We are now switching back to a single-layer NN, but with a greater number of nodes. We are going to call this the `wider_model`. Use a number of nodes that is twice as large as the number of inputs.

Evaluate the performance of the new estimator. Which of the three architectures gives you the best performance?

In this last exercise, we explored several modifications in the original model (standardised inputs, denser and wider architectures) in order to get a feel about the potential impact of architectural design changes on the NN performance. Having now some experience about what works (and what not so well), we are ready to tackle the last task.

Exercise #5 (3 points):

(a) **(2 points)** Use-your-creativity problem: Try to use a slightly different network to improve the average R^2 score. For example, you could try changing the number of layers, and/or the number of nodes, and/or the number of input features if you are feeling brave. You get two full points if you beat my score (I got $R^2 = 0.82 \pm 0.01$).

(b) **(1 point)** Plot the actual-vs-predicted scatter plot for the best σ_p/p model. Obtain the predicted values with the following call:

```
from sklearn.model_selection import cross_val_predict
predicted = cross_val_predict(pipeline, data, target, cv=kfold)
```

Then, create a $(\sigma_p/p)_{\text{true}}$-vs-$(\sigma_p/p)_{\text{pred}}$ scatter plot and overlay a $y = x$ diagonal line to see how well you have done.

Finally, reproduce the scatter plot of Fig. 1 by using the predicted values for the relative muon resolution, $(\sigma_p/p)_{\text{pred}}$.

6 Summary

In today's lecture we have seen how to employ regression with Neural Networks in order to make a quantitative prediction of a continuous variable based on a number of input features. This is very similar to the good ol' fitting problems that physicists have a lot of experience with. We have seen how changes in the architectures (*e.g.* in the number of nodes or layers of the NN) can affect significantly the performance of the estimator.

There is not a general rule on best practice for tackling this type of problems, *i.e.* on whether a deeper or a wider NN is the best approach. Empirical testing is usually the best approach for developing highly-performant NN for regression problems.

References

[1] The LHCb collaboration, `http://lhcb.web.cern.ch/lhcb/`

[2] "χ_{c_1} and χ_{c_2} resonance parameters with the decays $\chi_{c_1} \to J/\psi\mu^+\mu^-$", Phys.Rev.Lett. 119 (2017) no.22, 221801, DOI: 10.1103/PhysRevLett.119.221801, arXiv:1709.04247

[3] "Generalization of the Gluckstern formulas II: Multiple scattering and non-zero dip angles", M. Valentan , M. Regler, R. Frühwirth, Nuclear Instruments and Methods in Physics Research A 606 (2009) 728–742, DOI: 10.1016/j.nima.2009.05.024

[4] "Particle Detectors", C. Grupen and B. Shwartz, Second Edition, Online ISBN: 9780511534966, DOI: 10.1017/CBO9780511534966

[5] `Keras`, François Chollet *et al*, `https://keras.io`

[6] `https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics`

[7] Some of the Keras options you can try out:

- For the available types of activations, see: `https://keras.io/activations/`
- For the available types of initialisations, see: `https://keras.io/initializers/`
- For the available types of loss functions, see: `https://keras.io/losses/`
- For the available types of optimisers, see: `https://keras.io/optimizers/`