Report

A study of machine learning algorithms for reconstruction of missing energy in particle physics experiments.

Max Isacson, `max.isacson@physics.uu.se`
Mikael Mårtensson, `mikael.martensson@physics.uu.se`
Camila Rangel Smith, `camila.rangel@physics.uu.se`
Henrik Öhman, `ohman@cern.ch`

April 21, 2016

1 Introduction and Objective

2 The Dataset

2.1 Structure

The dataset consists of simulated pp collision events, in which a charged Higgs is produced and decays as $H^+ \to \tau\nu$. Each event is described by one set of observable variables and one set of unobservable variables.

Observable variables:

- E_x^{miss}, E_y^{miss}— The x- and y-components of the missing energy.

- $P_{\tau_{\mathrm{vis.}}}$ — The 4-momentum of the visible (hadronic) part of the τ decay.

- P_{b_0}, P_{b_1}, P_{q_0}, P_{q_1} — The 4-momenta of the two b-jets and two light jets.

Unobservable variables:

- P_{ν_τ} — The 4-momentum of the neutrino from the charged Higgs decay.

- $P_{\bar{\nu}_\tau}$ — The 4-momentum of the neutrino from the τ decay.

2.2 Production

MG5_aMC@NLO [1] is used for the matrix element computation of $gg/q\bar{q} \to H^+$ and the event simulation. The events are then passed to PYTHIA8 [2] for the showering and hadronization, and for the $H^+ \to \tau\nu$ decay. Finally, the detector response is simulated using DELPHES [3] with an ATLAS-like geometry.

2.3 Estimated and true quantities

The true quantities for both the observable and the unobservable variables are available as output from the event simulation. The estimated values for the observable variables are reconstructed from the output from the detector response simulation.

3 Methods and Implementation

3.1 Predictor selection

3.2 Artificial Neural Network

The data needs to be scaled to avoid the tails of the Artificial Neural Network (ANN) activation functions and improve the learning rate. The method used here is to scale it such that the minimum is -1 and the maximum is 1 for each training set variable. The test set data is scaled using the scaling parameters computed from the training set. Another scaling method would be to scale the mean to 0 and the standard deviation to 1, but this was found to slow down the training and give worst result.

The ANN implemented here uses Python with SciPy [4], Numpy [5], and Pandas [6] for general data processing, and Keras [7] for the actual neural network. The number of input nodes is restricted to the 19 selected predictor variables and the output to the target dimension of 1. Since the network is used for regression, the activation functions between the last hidden layer and the output must be linear.

Training is performed using stochastic gradient descent with the Nesterov method [8] and a mean-squared-error loss function. The learning rate was set to 0.1, the learning rate decay to 1×10^{-6}, and the momentum to 0.9.

A large number of configurations of the hidden layers were tested by varying the number of layers, the number of perceptrons in each layer, and the activation functions (sigmoid, tanh, and softmax). Sigmoid activation functions produced the best result for a fixed training period. A network with two hidden layers (i.e. 4 layers counting the 19 node input layer and 1 node output layer) with 25 and 10 perceptrons was found to be sufficiently complex. Using deeper networks, e.g. one with 4 hidden layer with 20, 35, 25, and 15 perceptrons, did not improve the result.

3.3 Support Vector Regression

Support vector machines can be used for regression, and the method is then called support vector regression (SVR). It employs two slack variables, $\xi_n \geq 0$ and $\hat{\xi}_n \geq 0$ and the corresponding error function to minimize is

$$C \sum_{n=1}^{N} (\xi_n + \hat{\xi}_n) + \frac{1}{2} ||w||^2$$

where C is the cost variable. The implementation used is the `SVR` class from `scikit-learn`, and it is an ε-SVR method. This implementation has four built-in kernels: RBF, sig-

moid, linear, and polynomial. The selected predictors in the dataset are scaled to have a mean of 0 and a variance of 1 using the `StandardScaler` class from `scikit-learn`.

With the presented dataset, the selected SVR algorithm is not converging within a few minutes runnning time. The maximum number of iterations is therefore limited to 1000. A greater number of iterations is not found to yield significantly better results. The kernel that performs best is the RBF kernel. The value of the cost function is by default set to 1.0, but this setting yields a very poor resolution. Different values of the cost function are tested, and a value of $C = 100.0$ is found to be (near) optimal. For the soft margin the default value of $\varepsilon = 0.1$ is good.

Regardless of the choice of C and ε, the performance of the SVR is found to be no better than that of linear ridge regression. The resolution is not improved, and the shape of the true and reconstructed transverse component of the missing energy is not recovered by the prediction.

3.4 Trees and forests

Decision trees and ensemble methods, and in particular boosted decision trees (BDTs), have been a popular choice of method in particle physics. The method of decision trees are based on using multiple discriminating variables, and multiple choices in a binary tree, to find the prediction that best describes the target variable. When using ensemble methods, such as extra trees, gradient boosting, or random forests multiple such trees are created, each with its own weight describing its "importance". The tree depth plays an important role when considering the trade-off between over-training and bias, and with the ensemble methods the number of trees is also an important parameter.

The performances of the classes `DecisionTreeRegressor`, `ExtraTreesRegressor`, `GradientBoostingRegressor`, and `RandomForestRegressor` in `scikit-learn` are examined. Different settings for the tree depth and (for the ensemble methods) number of trees are used. Near optimal values for the depth and the number of trees is found to be 15 and 100 for all methods and all ensemble methods respectively. The decision tree regression shows some improvement in resolution, but the shape of the transverse component of the missing energy is not recovered by the prediction. Neither in the ensemble methods is the shape of the missing energy recovered, but the resolution is improved. All three ensemble method show similar performance.

4 Results

5 Discussion

Out of all the Machine Learning (ML) methods tried, ANN showed the best performance in terms of resolution and predicting the p_T of the tau neutrino.

This project focused on predicting the transverse momentum p_T of the tau neutrino ν_τ by using the reconstructed four-momenta and missing energy seen by the detector as input and training on the true p_T given by the Monte-Carlo simulation. In the end it

is the mass of the H^+ that is interesting, which means that one option is to target the H^+ mass directly. This would require Monte-Carlo for a range of possible H^+ masses (3 mass-points were used in this study). Another option is to target the complete three-vector of the missing energy or the (combined or separate) four-vectors of the two neutrinos. These options were briefly looked at but were considered to be out of the scope of this project. They will be pursued in more detail in future studies.

To compute the missing transverse mass M_T (described in sec. 4) from the p_T predicted by the ML methods, the angle ϕ of the p_T is needed. However, since we are only predicting the p_T, the angle ϕ is unknown and is instead taken from the angle of M_T. It would have been better to also predict ϕ. This will also be studied in the future.

References

[1] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H. S. Shao, T. Stelzer, P. Torrielli, and M. Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *JHEP*, 07:079, 2014.

[2] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O. Rasmussen, and Peter Z. Skands. An introduction to PYTHIA 8.2. *Computer Physics Communications*, 191:159 – 177, 2015.

[3] J. Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi. Delphes 3: a modular framework for fast simulation of a generic collider experiment. *Journal of High Energy Physics*, 2014(2):1–26, 2014.

[4] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2016-04-19].

[5] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[6] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[7] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[8] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). *Soviet Mathematics Doklady*, 27(2):372–376, February 1983.