

M1 Internship at CERN

Mathis Reymond

May - July 2023

Contents

| | | |
|----------|---|----------|
| 1 | Introduction | 2 |
| 1.1 | Reading remarks | 2 |
| 1.2 | Useful links | 2 |
| 1.3 | Background and motivation | 2 |
| 1.4 | Challenge's setup | 2 |
| 2 | Domain Adversarial Neural Network | 2 |
| 2.1 | Prior art : pivot implementation | 2 |
| 2.2 | Two-branched implementation | 3 |
| 2.3 | Pivot vs two-branched | 4 |
| 2.4 | Further work | 5 |
| 2.4.1 | Architecture | 5 |
| 2.4.2 | Model's size | 5 |
| 2.4.3 | Further comparisons | 6 |
| 2.4.4 | Shrinkage impact and potential remedy | 6 |
| 3 | Toy Challenge | 6 |
| 3.1 | Simulator | 6 |
| 3.2 | Evaluation | 6 |
| 3.3 | Baselines | 7 |
| 3.3.1 | Two-branched results | 7 |
| 3.3.2 | 3D two-branched | 7 |
| 3.4 | Method and results | 7 |
| 3.4.1 | Flaws | 7 |
| 3.4.2 | Further work | 8 |
| 4 | Physics Update | 9 |
| 4.1 | Simulator | 9 |
| 4.2 | Evaluation | 10 |
| 4.3 | Baseline | 10 |
| 4.3.1 | Method | 10 |
| 4.3.2 | Results | 10 |
| 4.3.3 | Quantification of bending problem | 10 |

1 Introduction

1.1 Reading remarks

Section 2 is independent from the context of the challenge. Section 4 is independent from Section 3, except for subsub-section 4.3.3.

1.2 Useful links

[Here](#) is the main GitHub repository of the challenge.

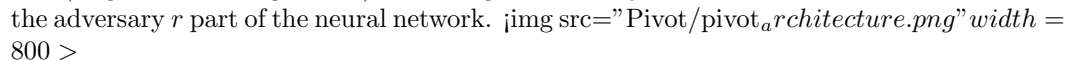
1.3 Background and motivation

1.4 Challenge's setup

2 Domain Adversarial Neural Network

This section is independent from the context of the challenge. We introduce and compare domain adversarial models, building on prior art. Domain adversarial models are designed to perform domain adaptation, that is, to address the task of transferring knowledge from a source domain to a target domain, where the distributions of the data differ. We will focus on two implementations of Domain Adversarial Neural Network (DANN) : pivot implementation and two-branched implementation. In the context of the challenge ([sections 3](#) and [4](#)) we only use the two-branched implementation.

2.1 Prior art : pivot implementation

In the paper, the implementation choosen for adversarial training is the pivot implementation, from the paper [Learning to Pivot with Adversarial Networks](<https://arxiv.org/pdf/1611.01046.pdf>), published in 2016. At the end of the classifier f is a layer with a single neuron which is used for class label prediction ("signal" or "background"). This single-neuron layer is then used as root for the adversary r part of the neural network.  `img src="Pivot/pivot_architecture.png" width = 800 >`

Philosophy : with this implementation, we want to train only one model and to use it for all test sets.

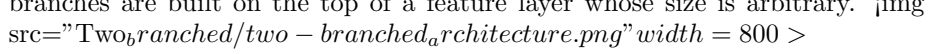
Pivot's usage pipeline :

0. You are given n test sets with unknow nuisance values
1. Sample nuisance values at regular interval from nuisance range
2. Simulate a train set with each nuisance value
3. Instantiate one model
4. Train the model providing - inputs (2D points) of each train set - outputs (class labels) of each train set - nuisance value of each train set
5. Predict class labels of each test set

`img src="Pivot/pivot_schema.png" width = 400 >`

Remarks : 1. You instantiate/train only one model 2. The model is efficient for predicting the labels of any test set 3. The nuisance values of the train sets are chosen to span the whole nuisance range. 4. The way nuisance values are used for training is not conflictual with the fact that the model doesn't know the nuisance values of the test sets. They are provided to the adversary part of the model, which is not involved in predicting the class labels after training.

2.2 Two-branched implementation

By comparison, another paper published in the same year, called [Domain-Adversarial Training of Neural Networks](https://arxiv.org/pdf/1505.07818.pdf), introduces another architecture where the network splits in two branches. One is used for predicting the class label ("signal" or "background") and the other one is used for predicting the domain label ("source" or "target"). The two branches are built on the top of a feature layer whose size is arbitrary.  `img src="Two_branched/two - branched_architecture.png" width = 800 >`

Philosophy : with this implementation, we want to train as many models as there are test sets.

Two-branched's usage pipeline :

0. You are given n test sets with unknown nuisance values 1. Simulate one train set with no nuisance 2. for i in range(n) - Instantiate one model - Train this model, providing - inputs (2D points) of the train set - outputs (class labels) of the train set - a source domain label for each sample of the train set - inputs (2D points) of the i-th test set - a target domain label for each sample of the test set - Predict class labels of the i-th test set

`img src="Two_branched/two - branched_schema.png" width = 400 >`

Remarks : 1. You instantiate/train n models 2. Each model is efficient only for predicting the class labels of the test set it has been trained on 3. Class labels of the test sets are not used for training.

In the traditional configuration of a classification problem, contestants would be given a train set with features and class labels (either signal or background), and a test set with features only. Then contestants would train a model using the train set and make predictions with the test set that will be evaluated by the organizer.

It consists of a feature extractor, a domain classifier, and a task-specific classifier. DANN utilizes a domain adversarial training procedure to learn domain-invariant features, enabling effective knowledge transfer and improved performance in the target domain.

On the other hand, when training a DANN, in addition to train set, we provide the test set (whose does not contains class labels) to the DANN. However, since it does not access the class labels of this test set, the DANN does not explicitly learn to classify signal from background on the test set. Given a point either from the train set or the test set, what it can learn is which set the points belong. Thus, in addition to the class label, we define a domain label : either "1" for training points or "0" for testing points.

And what we want is the class classifier to classify the training points and testing points in the same way. This means we want the output layer of feature extractor to encode information about the points independently from the set they come from. To do so, the retropropagation of the domain classifier is classic : this branch learns to classify point between "train" and "test", but before retropropagating through the feature extractor, we apply a Gradient Reversal Layer whose effect is make the feature extractor unlearn the domain of the points instead of learning it.

So during the training, each point from the train set propagates through the domain classifier and the class classifier, but each point from the test set only propagates through the domain classifier (the class classifier computes no loss, the parameters from the class classifier are not updated and the parameters from the feature extractor and the domain classifier are affected just as if there was no class classifier).

To sum up : DANN does not learn to classify events from the target set. It learns to bend the decision boundary in the direction of the target set, and, sometimes, under certain circumstances, this bending will result in classifying target events reasonably well.

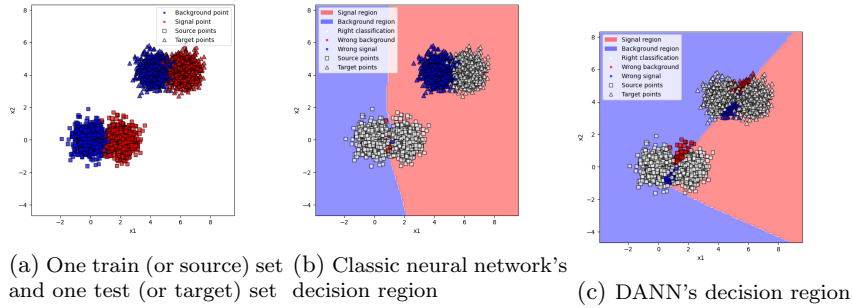


Figure 1: Classic neural network versus DANN behavior

BATCHES

Batch size is fix as an hyper parameter. Then the number of batches is variable, depending on the size of the simulated source and target datasets. So far, we suppose source and target simulated datasets to have the same size. Each training batch is half made of source events and half made of target events. Testing batches are fully made of target events.

2.3 Pivot vs two-branched

We are suggesting that the way the adversarial model is trained in the paper is inhibiting its core principle to operate on the Gaussian data used. Our remarks only rely on the way the model is trained (whatever the metric is). We provide some quantitative measures of performances of the models (AUC and accuracy), but the qualitative analysis of adversarial models behaviour is MUCH MORE

RELEVANT.

- A "nominal set" is a dataset generated with $\frac{\pi}{4}$ as nuisance parameter. -
An "upper set" is a dataset generated with the highest value of the nuisance
parameter : $\frac{\pi}{2}$.

We sample 21 values from $[0, \frac{\pi}{2}]$ to be nuisance parameters of 21 train set.
We generate 40000 points with the simulator for each value.

PIVOT

Here we use the pivot implementation. We train it as it's done in the paper
: - We train with 75- We test it on the remaining 25

Both of these plots show the decision boundary of the pivot model. In the
region shaded in red, the model would classify the points as the signal. In the
region shaded in blue, the model would classify the points as the background.
The color of the points is their true color (independent of how the model classify
them). On the first plot

TWO-BRANCHED

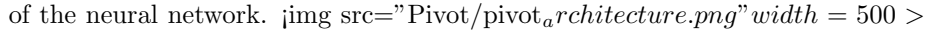
Here we use the two-branched implementation. - We train it with 75- We
test it with the remaining 25

the train-test split is the same as in 3.1

The model is trained with 21 times fewer points (only the points generated
with nominal values of the bias). The test points are exactly the same

2.4 Further work

2.4.1 Architecture

In the pivot implementation, at the end of the classifier f is a layer with
a single neuron which is used for class label prediction ("signal" or "back-
ground"). This single-neuron layer is also used as root for the adversary r part
of the neural network. 

*By comparison, the adversary (domain classifier) in the two-branched implementation is not built on top of the
imgsrc = "Two_branched/two-branched_architecture.png" width = 500 >*

*Intuitively, it seems natural to think that the shrinkage in the pivot implementation can impact negatively the perfor-
dimensional vector space. In this scenario, the model's output would be a vector of size n, and we would interpret them*

2.4.2 Model's size

Implementing the two-branched approach in the context of physics simulation
may not initially seem appealing, as it necessitates training a new model for
each simulation run. However, it is important to note that in this toy-context,
the two-branched model consists of only a few hundred parameters, whereas
the pivot implementation involves about one hundred thousand of parameters.
Therefore, it would be worthwhile to assess whether the computational cost
of the two-branched implementation remains manageable in a realistic physics
scenario. By conducting such an evaluation, we could determine if the benefits
of reduced parameter complexity outweigh the potential drawbacks of training
the model for each simulation, thus making it a feasible option for practical
applications in physics simulations.

2.4.3 Further comparisons

2.4.4 Shrinkage impact and potential remedy

3 Toy Challenge

3.1 Simulator

In the context of the toy challenge, we built a simple 2D simulator. Running the simulator one time produce a dataset, that is, a collection of 2D points with associated labels, either 0 for "background" class or 1 for signal class.

To simulate a dataset, we apply the following pipeline :

1. Provide input parameters to the simulator such as the total number of points to simulate N , the proportion p_b of background points to be in the output dataset, the nuisance values and distribution types (Gaussian, Poisson).
2. Define random variables B and S following a 2D probability law, one for background and one for signal.
3. Instantiate an empty dataframe with columns x_1 and x_2 to store the points and instantiate an empty list for their associated labels.
4. Get Np_b outcomes of B and store them in the Np_b first lines of the dataframe. Add Np_b times the label "0" to the labels list. In practise, p_b is such that Np_b is an integer.
5. Get $N(1 - p_b)$ outcomes of S and store them in the dataframe after the background points. Add $N(1 - p_b)$ times the label "1" to the labels list, after background labels.
6. Shuffle dataframe's lines and labels in parallel.

3.2 Evaluation

Contestants, are given the simulator so they can generate as much data (both 2D points and labels) as they want for training. Especially, they are able to introduce the nuisance values they want to generate training data.

They are asked to predict the labels of 2D points from tests sets generated by the organizers with the same simulator. In this framework, contestants don't have access to :

- class labels of test data points
- nuisance values used by organizers to generate test sets

3.3 Baselines

3.3.1 Two-branched results

3.3.2 3D two-branched

In several cases, we observed that DANN can not propagate the decision boundary in the direction of target domain because of geometrical constraints. What would be beneficial for the model would be to have an extra dimension to draw the decision boundary.

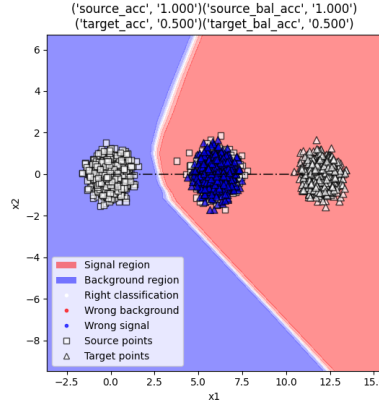


Figure 2: In this case, signal source points overlap with background target points, making it harder for the DANN to classify them accurately

3.4 Method and results

What you want to do is to send all target points away from the source points on a third axis. And you can do this

3.4.1 Flaws

- Instability - Geometrical propagation to target set - Bending We observed that DANN's behaviour was not symmetrical in that sense that in the case of a translation nuisance with signal and background horizontally aligned, it was harder for it to propagate decision boundary to a target set at its right than at its left PIC. Then we tried to set the λ parameter to 0, which is supposed to annihilate the adversarial behaviour, rendering DANN to be a simple neural network with no learning on the target set. In such a context, we would have expected the decision boundary to be a vertical line, but it is not. The shape of the decision boundary is a parabola bent toward the right and, which is more problematic, it is a constantly bent toward the right. The reason for the shape is the use of relu activation function, using hyperbolic tangent produce another curve for the decision boundary PIC. Moreover, using a linear activation produce a line for

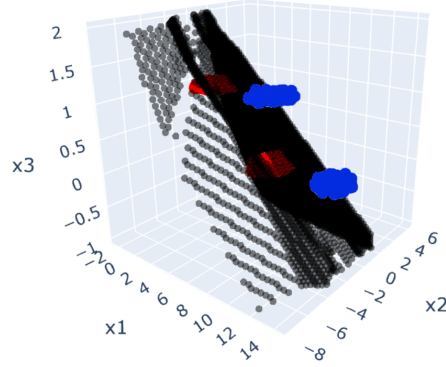


Figure 3: Points of the source set are at 1 on x_3 while points from target set are at 0. All signal points are below the decision boundary and all background points are above it : the DANN accurately classify the points

the decision boundary, however, it is never vertical but persistently tilted in the same direction PIC. The reason for this shift in the decision boundary despite the symmetrical appearance of the two distributions is that, as opposed to other methods, neural networks takes absolute position as input, not only relative position of the points. Thus, if the relative position of the points remains the same but they are sent far from the origin, the model would perform poorly to classify them PIC. Then, the learning on the source set could be improved by adding more complexity to the model, which would results to a better classification on the source set. However, outside the region of the source points, the decision boundary remains uncontrolled, which is a problem to classify target points as the domain adaptation would be impacted. The point is that when the lambda parameter is not nul, the DANN behaves as a regular neural network biased in a favorable way by the adversary ; thus, the imperfections of the behaviour of a regular neural network will reverberate on DANN's behaviour.

Fortunately, in a realistic scenario, the impact of this issue is lessened by two factors :

1. the nuisance magnitude is very small. If the nuisance was a translation, you would not be able to determine the translation direction by looking at the points.
2. physicist apply preprocessing techniques to prevent this kind of issues.

See [section 3](#) for a quantitative evaluation of the problem in a more realistic scenario.

3.4.2 Further work

- Coping with instability - Coping with bending problem - 3D DANN : distance on the third axis - nD two-branched - Play with feature space - Improve domain classifier - Machine learning class label classifier

4 Physics Update

4.1 Simulator

A real physics simulator generates a lot of events, most of which is obvious background events. Thus, a first step is to get rid off these events. After this filtering, we end up with fewer background events, but they are still far more numerous than signal events (two order of magnitude).

To simulate the simulation We represent the number of background events by a random variable which is assumed, from previous experiments, to follow a Poisson distribution with β parameter. We assume, from theoretical knowledge, that we can represent the number of signal events by a random variable following a Poisson distribution with γ parameter. We define $\nu = \beta + \mu\gamma$ the random variable that represent the total number of events, where μ is a random variable following a continuous uniform law of parameters 0.9 and 1.1. This μ parameter represents how much the expected number signal events simulated deviates from what is claimed by theory.

To simulate a dataset, we apply the following pipeline :

1. we arbitrarily set π , the expected balanced between background and signal events as stated by theory.
2. we arbitrarily set ν_1 , the expected number of events if $\mu = 1$ (it gives the order of magnitude of the size of the dataset).
3. we compute the expected number of background events, $\beta = \nu_1\pi$
4. we compute the expected number of signal events stated by theory, $\gamma = \nu_1(1 - \pi)$
5. we sample one value of μ
6. we compute $\nu = \nu_1(\mu\pi + (1 - \pi))$ the expected number of event of the dataset.
7. we compute $p_b = (1 - \pi)/(\mu\pi + (1 - \pi))$ the proportion of signal events in the dataset.
8. we compute $p_s = \pi\mu/(\mu\pi + (1 - \pi))$ the proportion of background events in the dataset.
9. we draw N from $\text{Poisson}(\nu)$ for the actual number of events in the dataset.
10. we draw z , the nuisance value, uniformly from a reasonable range
11. we generate Np_b signal events and Np_s background events, each of which is biased by z .

Thus, each simulated dataset has a specific nuisance value, a specific number of events and a specific background to signal ratio.

Information contestants don't have access to :

- μ value of test sets
- class label of test sets events
- z value of test sets

4.2 Evaluation

In this configuration, contestants are asked to estimate μ .

4.3 Baseline

4.3.1 Method

4.3.2 Results

4.3.3 Quantification of bending problem

5 Conclusion

[3] [2] [1] [4]

References

- [1] Claire Adam-Bourdarios et al. "The Higgs boson machine learning challenge". In: *Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning*. Ed. by Glen Cowan et al. Vol. 42. Proceedings of Machine Learning Research. Montreal, Canada: PMLR, Dec. 2015, pp. 19–55. URL: <https://proceedings.mlr.press/v42/cowa14.html>.
- [2] Yaroslav Ganin et al. *Domain-Adversarial Training of Neural Networks*. 2016. arXiv: [1505.07818](https://arxiv.org/abs/1505.07818) [stat.ML].
- [3] Aishik Ghosh, Benjamin Nachman, and Daniel Whiteson. "Uncertainty-aware machine learning for high energy physics". In: *Physical Review D* 104.5 (Sept. 2021). DOI: [10.1103/physrevd.104.056026](https://doi.org/10.1103/physrevd.104.056026). URL: <https://doi.org/10.1103/physrevd.104.056026>.
- [4] Gilles Louppe, Michael Kagan, and Kyle Cranmer. *Learning to Pivot with Adversarial Networks*. 2017. arXiv: [1611.01046](https://arxiv.org/abs/1611.01046) [stat.ML].