M1 Internship at CERN

Mathis Reymond

May - July 2023

Contents

1 Introduction

1.1 Reading remarks

Section 2 is independent from sections 3 and 4 except for sub-section 2.2. 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

Let's 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 classification 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

They are several implementations for domain adversarial neural networks. Pivot implementation is one of them. It has been proposed in [4] and applied to overcome systematic uncertainty in physics data in [3]. See 1 for visualization of the implementation.

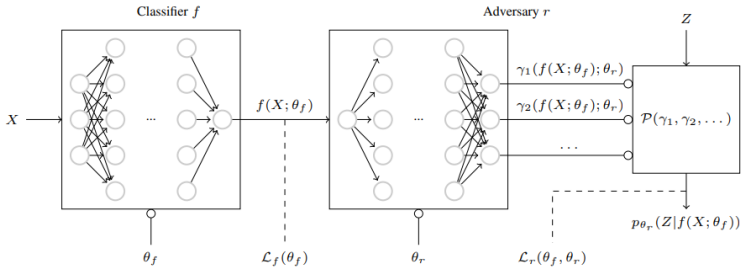


Figure 1: Pivot model's architecture

In this implementation are two components : the Classifier denoted as f and the Adversary denoted as r. f is meant to label classification. At the end of this first component is a layer with a single neuron which is used for label prediction ("signal" or "background"). This single-neuron layer is also used as input to r. r has another input Z, the value of the nuisance parameter.
To train this model, one has to generate several train sets with specific nuisance values Z, each of which correspond to one specific domain. The nuisance values used for generating these train sets have to be chosen so they span the whole nuisance range. During training, the pivot model learns label classification on domains that are a discrete approximation of all possible domains and learns to encode the input x in the first layers of f indiscriminately with respect to the domain. Once training is done, only the component f is used for predicting labels. When provided new data from an unknown domain, the component f already saw data from a close domain and knows how to encode the new data to perform domain-independent label classification.
Remarks :

1. This method involves training a single model capable of performing label classification on any test set.

2. The nuisance values of the train sets are intentionally selected to cover the entire range of nuisance.

3. The approach for utilizing nuisance values during training does not create any conflict with the fact that the nuisance values of the test sets are unknown. These values are supplied to the adversary component of the model, which is not engaged in predicting the labels after the training process.

2.2 Two-branched implementation

By comparison, [2] introduces another architecture where the network splits in two branches. Such a neural network is made of three components.

1. Feature extractor : This component is responsible for extracting meaningful features from the input data. The output layer of the feature extractor is shared between the label classifier and the domain adversary to learn domain-invariant representations of the data.

2. Label classifier : The label classifier responsible for the primary task the model is designed for. It takes the learned features from the feature extractor and predicts the labels of the input data ("signal" or "background"). The label classifier is trained to minimize the standard task-specific loss to make accurate predictions on the source domain only, as the labels of the target data is unknown.

3. Domain classifier : The domain classifier takes the learned features from the feature extractor and tries to predict the domain of the input data

("source" or "target"). The objective of the domain classifier is to maximize the domain classification accuracy while back-propagating to the feature extractor in such a way that confuses the domain classifier, making it difficult to determine the domain from the extracted features.

See 2 for visualization of the implementation.

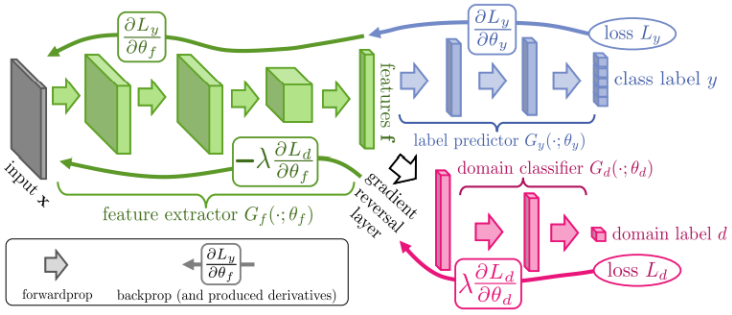


Figure 2: Two-branched model's architecture

DANN's training procedure requires to jointly train the label classifier, feature extractor, and domain classifier. The overall objective function during training can be formulated as follows:

$$\mathcal{L}_{total} = \mathcal{L}_{label} - \lambda\mathcal{L}_{domain} \tag{1}$$

Where:

- $\mathcal{L}_{label}$ is the label classification loss between the predictions of the label classifier and the true class labels ("signal" or "background")
- $\mathcal{L}_{domain}$ is the domain classification loss between the predictions of the domain classifier and the true domain labels ("source" or "target")
- λ is a positive real number. It is a hyper-parameter that controls the importance of the domain classification loss in relation to the label classification loss.

The adversarial learning is induced by the minus sign in $\mathcal{L}_{total}$ expression. It leads the feature extractor to minimize the domain classification accuracy while the domain classifier learns to maximize it.
Training the two-branched model requires to generate a source set with no bias and the points of one test set, referred as target set. Source and target points are merged. When the model is learning from a source sample, the forward and back propagation happen both in the label classifier and the domain classifier. When training the model on a target sample, the forward and back propagation happen only in the domain classifier. As the label classifier and domain classifier learn to

maximize label classification accuracy and domain classification accuracy, the feature extractor learns to minimize the domain classification accuracy while maximizing label classification accuracy. It is this adversarial objective between feature extractor and domain classifier that leads to domain-invariant extracted features at the end of the training, allowing the label classifier to predict the labels of source and target data indiscriminately. Remarks :

1. This method requires training as many models as there are test sets.
2. Each model capable of performing label classification on the test set used to train it.
3. The nuisance values of the train sets are intentionally selected to cover the entire range of nuisance.
4. The approach for utilizing test points during training does not create any conflict with the fact that the test labels are unknown.

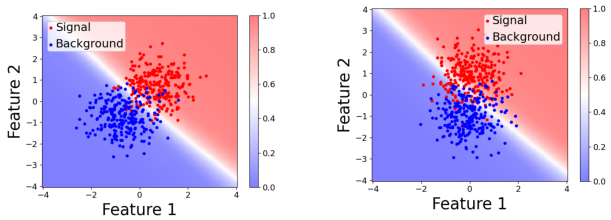
Batch size is set as an hyper parameter. The number of batches depends on the number of epochs and the size of the source and target sets. Each training batch is half made of source samples and half made of target samples. Testing batches are fully made of target events.

2.3 Comparison

Two objections can be raised to the pivot implementation :

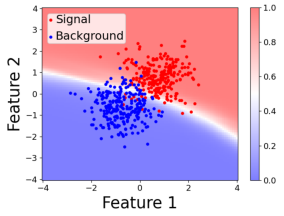
1. At the end of the classifier f is a single-neuron layer on the top of which is build the adversary r. The impact of this shrinkage on the entire learning process has not been discussed yet. Contrarily, in the two-branched implementation, the domain classifier is built on the top of the feature f layer whose size is arbitrary. The implications of this difference in architecture on the learning process remain unexplored.
2. In the physics framework, the model has to predict the label ("signal" or "background") of actual LHC's events. However, all these events belong to the same domain and constitute one single test set. Hence, there might be no need to train a model to be compliant with any domain if it is possible to train a model specifically to be compliant with the test events' particular domain.

Moreover, adapting to multiple domains simultaneously can result in conflicting learning outcomes. In the context of the section discussed in [3], it leads to no adaptation at all. The systematic perturbation introduced in this context is a rotation angle, with a nominal value of $\frac{\pi}{4}$ and a nuisance value z within the range $[-\frac{\pi}{4}, \frac{\pi}{4}]$. The test (or target) set is generated with the angle set to $\frac{\pi}{2}$.
To train the pivot model, the authors generated 21 datasets with angles ranging from 0 to $\frac{20}{20}\frac{\pi}{2}$ in increments of $\frac{1}{20}\frac{\pi}{2}$. This implies that the pivot model attempts to adapt to angles greater and smaller than $\frac{\pi}{4}$ simultaneously. Consequently,

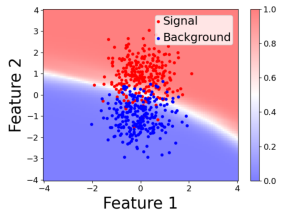


(a) Pivot boundary with source points. 97% accuracy.

(b) Pivot boundary with target points. 91% accuracy.



(c) Two-branched boundary with source points. 96% accuracy.



(d) Two-branched boundary with target points. 96% accuracy.

Figure 3: Decision boundaries of pivot and two-branched models

the decision boundary becomes optimal for an angle of $\frac{\pi}{4}$, which is equivalent to what a plain neural network would achieve. Consequently, the model performs exceptionally well on the nominal set (see 3a), which is uninteresting, but exhibits poor performance on the target set (see 3b).
In contrast, during training, the two-branched model is provided with a source set comprising the nominal value of the angle and the points from the target set. Thus, during the learning process, the model makes a trade-off between source and target domains. As a result, it may perform slightly worse on the source set (see 3c), which is not problematic since the performance on the source set is of no interest. However, it performs significantly better on the target set (see 3d).

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 "background"). This single-neuron layer is also used as root for the adversary r part of the neural network. ¡img src="Pivot/pivot$_architecture.png"width = 500 >
$By comparison, the adversary (domain classifier) in the two-branched implementation is not built on the top of the$
$imgsrc = "Two_branched/two - branched_architecture.png"width = 500 >$
$Intuitively, it seems natural to think that the shrinkage in the pivot implementation can impact negatively the per fo$

$dimensional vector space. In this scenario, the model's output would be a vector of size n, and we would interpret then$

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 :

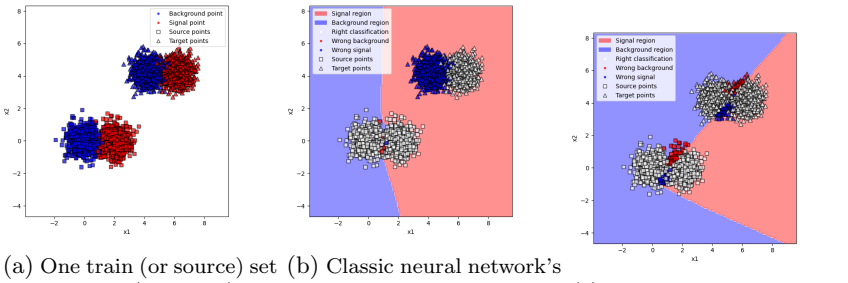
- class labels of test data points
- nuisance values used by organizers to generate test sets

3.3 Baselines

3.3.1 Two-branched results

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.



(a) One train (or source) set and one test (or target) set

(b) Classic neural network's decision region

(c) DANN's decision region

Figure 4: Classic neural network versus DANN behavior

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.

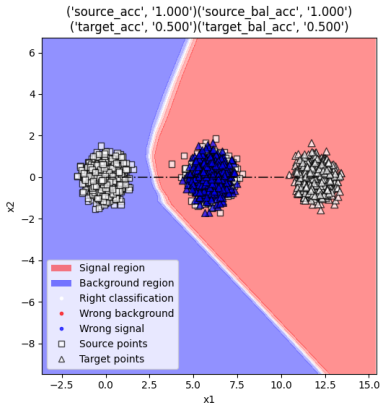


Figure 5: In this case, signal source points overlap with background target points, making it harder for the DANN to classify them accurately

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

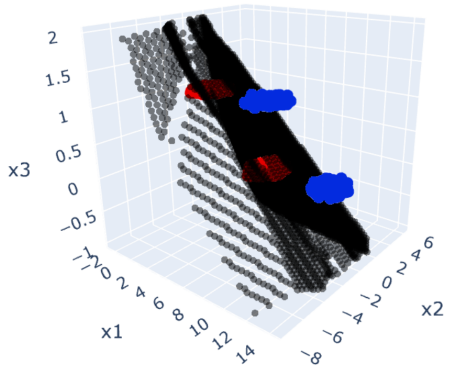


Figure 6: Points of the source set are at 1 on x_3 while points from target set are at 0.All signal points are below the decision boundary and all background points are above it : the DANN accurately classify the points

3.4.1 Flaws

- Instability - Geometrical propagation to target set - Bending We observed that DANN's behaviour was not symmetrical in that sens that in the case of a translation nuisance with signal and background horizontaly aligned, it was harder for it to propagate decision boundary to a target set at his right than at his left PIC. Then we tried to set the λ parameter to 0, which is supposed to anihilate

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 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 the decision boundary, however, it is never vertical but persistently titled in the same direction PIC. The reason for this shift in the decision boundary despite the symetrical 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 send 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 compexity 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 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 [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. URL: https://doi.org/10.1103%2Fphysrevd.104.056026.

[4] Gilles Louppe, Michael Kagan, and Kyle Cranmer. *Learning to Pivot with Adversarial Networks*. 2017. arXiv: 1611.01046 [stat.ML].