Fair-Universe Toy-Challenge Report

Mathis REYMOND,Stefano BAVARO, Jacobo RUIZ, Shah Fahad HOSSAIN

March 2023

Challenge page - https://www.codabench.org/competitions/565/?secret_key=35329465-c378-4483-9564-8a4a4bf617ba#/pages-tab

1 Background and motivation

High energy physicists at CERN use simulations in order to reproduce collisions that occurs in the LHC. Collisions between particles create thousands of smaller particles. Once they developed a theory which predicts the existence of a new particle, physicists run these simulations and seek for evidences of the new particle. To do so, they classify all the particles resulting from a collision between background particles (uninteresting ones that they already know) and signal particles (the ones they are interested in). This is why high-energy physicists are working increasingly closely with machine learning scientists. To perform this classifications task there are tens of available features about each particle (such as its speed, its energy or some angles measurements). However, the simulations are prone to systematic biases and so it is for the data used for classification, which makes the task harder. Thus, a big challenge is to remove this biases from the data in order to improve the accuracy of classification. The Fair Universe challenge is a toy-example for this problem. Instead of working in a high-dimensional feature space, we represent the particles as 2D points that belongs either to signal or background class. The aim is to build a model that classifies them correctly. We represent the biases that affects the particles as a translations that affects all the points (with no regard to their class).

2 Data and task description

2.1 Task

In this challenge, in order to better capture physicists problem, we do not ask contestants to solve one task, but several ones. We provide several training sets. Each training set corresponds to a specific task we want contestants to tackle,

that is the points have been generated in a specific way. The points in training sets have no bias. For each training set, we provide several test sets that have been generated in the same way, but the points in these test sets suffer from bias. We want contestants to tackle all the tasks and for each of them, to train a model able to overcome the bias and classify the points no matter the direction and magnitude of the translation.

2.2 Parameters

Before describing the datasets provided, let's introduce the parameters used to generate them. Signal and background points are generated independently using 2D-Gaussian distributions.

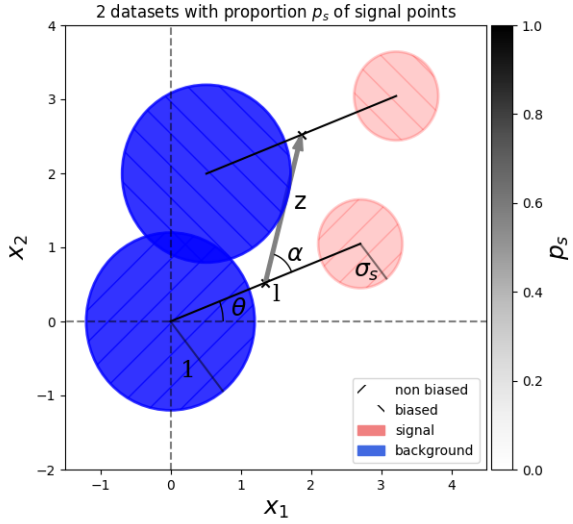

Figure 1: Illustration of all the parameters that defines a dataset's task. Each filled circle represents a Gaussian distribution. The dotted circle is centered in (0,0) and indicates the center of non-biased distributions.

Here are the parameters specifications. See Figure 1 for illustration.

- $p_s \in]0, 0.5]$: the proportion of signal points
- $\sigma_s \in]0, 1]$: standard deviation of signal distribution
- $\theta \in [0, 2\pi[$: angle that indicates the direction of distributions alignment
- $l \in]0, \infty[$: distance between distribution's centers
- $\alpha \in [0, 2\pi[$: angle that indicates the direction of the translation when added to θ

- $z \in [0, 5]$: nuisance value, magnitude of the translation

Background distribution is always centered at $(0, 0)$ and its standard deviation is always set to 1.

1. For each train set, we fix the parameters p_s, σ_s, θ, l and we generate 1000 points.

2. For each test set, we fix the parameters p_s, σ_s, θ, l, we generate 1000 points. Then we fix α and z and we apply to all the generated points the translation in the direction $\theta + \alpha$ with magnitude z (in polar coordinates).

2.2.1 Mathematical formalism

Here is a more detailed mathematical formalism for the way we generate the data. See again Figure 1 for illustration.

Let a *task* be a tuple $(p_s, \sigma_s, \theta, l)$. A *context* is a tuple (σ, θ, l). Given a context $C = (\sigma, \theta, l)$, a *point generated in the context* C is the realization of a random variable that follows a bivariate Gaussian law of parameters $(l\cos\theta, l\sin\theta)$ and $\begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix}$.
Given θ, a *nuisance direction* α and a *nuisance magnitude* z, let

$$\tau_{\theta,\alpha,z} : (\hat{x_1}, \hat{x_2}) \in R^2 \mapsto (x_1, x_2) := (\hat{x_1} + z\cos(\theta + \alpha), \hat{x_2} + z\sin(\theta + \alpha)).$$

Given a context $C = (\sigma, \theta, l)$, a nuisance direction α, a nuisance value z and a *label* $c \in \{0, 1\}$, a *c-labelled event for z in the direction α in the context C* is a tuple (x_1, x_2, c), where (x_1, x_2) is the image by $\tau_{\theta,\alpha,z}$ of a point generated in the context of C.
Given a task $T = (p_s, \sigma_s, \theta, l)$, a nuisance direction α and a nuisance value z, a *dataset for T plagued by α and z* is a set containing $1000p_s$ 1-labelled events for z in the direction α in the context (σ_s, θ, l) and $1000(1-p)$ 0-labelled events for z in the direction α in the context $\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \theta, 0\right)$.
For each labelled event (x_1, x_2, c), we define its associated *event* (x_1, x_2).
The aim of this challenge is, for any given task T, to build a machine learning model that most accurately predicts the class of events no matter what nuisance direction and value were used to generate them. To do so, contestants are provided one train dataset for T plagued by 0 and 0.

2.3 Data

As it would have been impossible to provide all possible combinations of the parameters presented in the previous section due to combinatorial explosion, we decided to define 16 representative tasks, creating 1 train dataset and 4 test datasets with different nuisance values for each. We kept fixed the number of

points per dataset to 1000, the center of the background distribution to $(0, 0)$ and its standard deviation to 1. Then the train sets are generated with 16 combinations of these parameters values :

- $p_s \in \{0.05, 0.5\}$
- $\sigma_s \in \{0.1, 1\}$
- $\theta \in \{0, \frac{\pi}{4}\}$
- $l \in \{1, 3\}$
- $(z, \alpha) = 0$

And the test sets are generated with 64 combinations of these parameters values :

- $p_s \in \{0.05, 0.5\}$
- $\sigma_s \in \{0.1, 1\}$
- $\theta \in \{0, \frac{\pi}{4}\}$
- $l \in \{1, 3\}$
- $(z, \alpha) \in \{(0, 0), (5, 0), (5, \frac{\pi}{4}), (5, \frac{\pi}{2})\}$

2.4 Metric

The metric used to evaluate the obtained result is the ROC AUC. In this section we give a comprehensive explanation of it.

2.4.1 ROC Curve

An ROC (Receiver Operating Characteristic) curve is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate (TPR) or Recall:

$$TPR = \frac{TP}{TP + FN}$$

- False Positive Rate (FPR):

$$FPR = \frac{FP}{FP + TN}$$

A ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. Figure 2 shows a typical ROC curve.

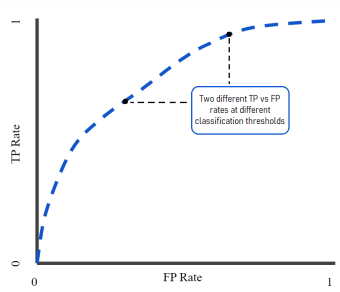

Figure 2: ROC Curve showing TPR vs. FPR at different classification thresholds

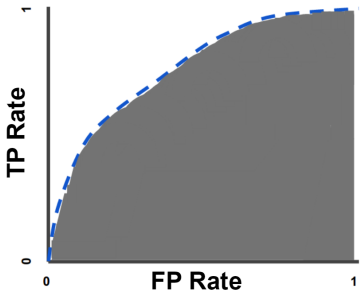

Figure 3: Area under the previous ROC Curve

2.4.2 ROC AUC

ROC AUC stands for "Area under the ROC Curve". That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). See 3.

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0. See [1] for more explanation.

3 Challenge Design

The problem is a binary (two classes) classification problem. Each sample (a 2D point) is characterized by its coordinates x1 and x2 (2 features). You must predict the points category: signal or background. We have given for training a data matrix X_train of dimension num_training_samples x 2 (2 features, x1 and x2) and an array y_train of labels of dimension num_training_samples. The participants must train a model which predicts the labels for two test matrices X_valid and X_test.

3.1 Evaluation

The challenge is designed into two phases.

Phase 1-Development Phase: We provide the participants with labeled training data and unlabeled validation and test data. Make predictions for both datasets. However, the participants will receive feed-back on your performance on the validation set only. The performance of their LAST submission will be

displayed on the leaderboard.

Phase 2-Final Phase: The participants do not need to do anything. Their last submission of phase 1 will be automatically forwarded. Their performance on the test set will appear on the leaderboard when the organizers finish checking the submissions.

This sample competition allows you to submit:
1. Only prediction results (no code).

3.2 Rules

1. Submissions must be made before the end of phase 1.
2. You may submit 5 submissions every day and 100 in total.

4 Baseline methods

4.1 Naive baselines

We came up with 4 naive baselines :

1. Constant prediction. We systematically classify the points as background.

2. Naive Bayes. We train a naive Bayes classifier on a train set. This method doesn't take the bias into account.

3. Data augmentation. Given a train set and a test set, we calculate the average coordinates of each set to retrieve the direction and magnitude of the nuisance. Then we generate an extra dataset using the training points translated in the same direction but with a smaller magnitude than the test set. Finally, we train a naive Bayes classifier using both the train set and the extra dataset.

4. Statistical preprocessing. We train a naive Bayes classifier on a train set. We calculate its average coordinates. Then, for a given test set, we calculate its average coordinates and translate all its points to center it like the train set. Then we classify the translated test points with the naive Bayes classifier. Note that this method is actually some kind of "hack" for the competition since it does not use machine learning to deal with bias but simply removes it using a statistical approximation.

4.2 Manifold Tangent Classifier (MTC)

The MTC is a pipeline: we first extract information from our data with an extractor and we then use a classification algorithm that uses this information in order to be more robust to small data variations.
All the credits go to publications [4]. This method uses a modified version of a

classical 2 layered neural network for binary classification.
The only thing modified is the back-propagation algorithm of the network, it is modified by adding a new regularization weight to the back-propagation equation. This new regularization weight is obtained as follows:
We use an Auto-Encoder (you can think of it as a feature extractor for now) to extract "tangent vectors" at every data point x. These tangents basically encode information about how this data point x in particular may vary (by some basic transformation like a translation or rotation). Then we use these learnt tangents in the regularization term to make the back-propagation algorithm more insensitive to data variance.
This method is interesting as in this challenge the variance is the bias introduced to the particle 2D coordinates. Therefore an algorithm that is insensitive to small variances is ideal. See the method's description for more in depth understanding.
Please refer to appendix A for more details.

4.3 Domain Adversarial Neural Networks

The second method we decided to consider as a baseline is the Domain Adversarial Neural Network. It is inspired on the well-established theory in the field of domain adaptation (the process of training a model on data from one source domain, and adapting or transferring the model to work well on a different target domain) suggesting that, for effective domain transfer to be achieved, predictions must be made based on features that cannot discriminate between the training (source) and test (target) domains. In the context of this problem, the domain transfer is to be achieved in order to still perform a good classification when data at training and test time come from similar but different distributions (with a nuisance value \ bias applied). In this method, this idea is implemented using a neural network architecture, trained on labeled data from the source domain and unlabeled data from the target domain. As the training progresses, the approach promotes the emergence of features that are (i) discriminative for the main learning task on the source domain and (ii) indiscriminate with respect to the shift between the domains. The method is based on the formulation defined in [2] and the code we used to implement it is an adaptation of what contained in [3].
See appendix B for more details.

5 Results

In this section we show the test results of our 6 baselines. See Figure 4. For each parameter, we sort the values between easy (green font), medium (orange font) and hard (red font). For example, the farther signal and background clusters are, the easier is the classification task. Thus, $l = 3.0$ is the easy case and $l = 1.0$ is the hard one.
For a given baseline, for a specific value of a specific parameter, we evaluate the

baseline on all the test sets where this parameter is set to this value. Then we average the resulting scores. For example, the number in the row "DANN,AUC score" and the column "l, easy", we averaged all the AUC scores of the DANN on the datasets where l was set to 3.0.
Moreover, we had expectations on the results. To show that we expected a baseline to score equally, better or worse on an easy value than on a hard value of a given parameter, we put a "=", "¿" or a "¡" symbol respectively in the corresponding prediction row. For example, we expected the naive Bayes classifier to perform better on datasets where $z = 0$ (no bias) than on those where $z = 5$; thus, we put a "¿" symbol in the row "Naive Bayes Classifier, Prediction" and column "z, easy".
Finally, we use green background in predictions rows if the AUC score results match our expectations, and red background otherwise. For example, naive Bayes classifier has a greater AUC score in average when $z = 0$ than when $z = 5$, this match our expectation, thus the box at row "Naive Bayes Classifier, Prediction" and column "z, easy" has green background. Note that there is a 0.05 tolerance for "=" predictions.

	Parameter		p		l		σ_s		θ			α		z
	Difficulty	easy	hard	easy	hard	easy	hard	easy	hard	easy	medium	hard	easy	hard
	Value	0.5	0.05	3.0	1.0	0.1	1.0	0.0	0.71	1.57	0.71	0.0	0.0	5.0
Model														
Constant classifier	Prediction	=		=		=		=		=	=		=	
	AUC score	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
	BA score	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
Naive Bayes Classifier	Prediction	>		>		>		=		>	>		>	
	AUC score	0.669575	0.631666	0.697016	0.604224	0.469438	0.831802	0.641541	0.659700	0.657076	0.532992	0.706207	0.936538	0.555315
	BA score	0.639969	0.605296	0.655015	0.590250	0.618008	0.627257	0.621247	0.624018	0.600013	0.525181	0.682668	0.867010	0.541173
Data augmentation	Prediction	>		>		>		=		>	>		>	
	AUC score	0.812669	0.798885	0.968813	0.642741	0.754156	0.857398	0.806505	0.805049	0.786226	0.757389	0.839746	0.936076	0.762344
	BA score	0.616563	0.600691	0.644618	0.572635	0.626602	0.590651	0.601518	0.615735	0.550217	0.514405	0.684942	0.867987	0.522173
DANN	Prediction	=		=		>		=		>	>		=	
	AUC score	0.831568	0.832079	0.896735	0.766911	0.896025	0.767621	0.859701	0.803945	0.903671	0.846959	0.788332	0.927571	0.799907
	BA score	0.765531	0.601776	0.768120	0.599187	0.707059	0.660248	0.687574	0.679734	0.730181	0.653632	0.675401	0.808961	0.641885
Tangent model	Prediction	>		>		=		=		=	=		>	
	AUC score	0.854883	0.421347	0.735112	0.541118	0.662306	0.613924	0.581467	0.694762	0.676851	0.581793	0.646908	0.710874	0.613862
	BA score	0.601344	0.499803	0.572209	0.528938	0.556771	0.544375	0.544521	0.556625	0.562500	0.499938	0.569928	0.643438	0.519618
Statistical preprocessing	Prediction	>		>		>		=		=	=		=	
	AUC score	0.669575	0.631666	0.697016	0.604224	0.469438	0.831802	0.641541	0.659700	0.657076	0.532992	0.706207	0.936538	0.555315
	BA score	0.608313	0.801809	0.763456	0.646666	0.718806	0.691316	0.705230	0.704891	0.699734	0.712141	0.704184	0.705852	0.704797

Figure 4: Results

We can draw some conclusions. Firstly, our implementation of the tangent method sometimes performs very poorly (average AUC score under 0.5). We didn't manage to find what cause these performance drops and it needs further investigation.
Furthermore, we were wrongly expecting the naive Bayes classifier and the data augmentation method to perform better with small values of σ_s. We presume the contrary happens because naive Bayes classifier draw wider signal regions when σ_s is large, making it more compliant to bias.

6 Conclusions and future work

In conclusion, the Fair-Universe Toy-Challenge provide a valuable opportunity for participants to apply their knowledge of machine learning to help solving a real-world problem. Our 2 elaborated baselines showcased the potential of AI in advancing particle physics research and provided insights into the current limitations of existing classification methods.

There are several directions to further explore to improve this competition.

1. Try to prevent from the statistical preprocessing "hack". One solution might be to define a fixed window for each task and, for each dataset associated with this task, to provide only the points that lye inside this window. Doing this, the statistical approximation of the bias would become harder and less accurate.

2. Increase the number of feature. Use more complex data, closer to physicists problem.

7 Bibliography

References

[1] "Classification: courbe ROC et AUC". In: (Apr. 2022). URL: %5Curl%7Bhttps://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=fr%7D.

[2] Yaroslav Ganin et al. "Domain-adversarial training of neural networks". In: *The journal of machine learning research* 17.1 (2016), pp. 2096–2030.

[3] Benjamin Planche and Eliot Andres. *Hands-On Computer Vision with TensorFlow 2: Leverage deep learning to create powerful image processing apps with TensorFlow 2.0 and Keras*. https://github.com/wikibook/dl-vision/blob/master/Chapter07/ch7_nb5_train_a_simple_domain_adversarial_network_(dann).ipynb. Packt Publishing Ltd, 2019.

[4] Patrice Y. Simard et al. "Transformation Invariance in Pattern Recognition – Tangent Distance and Tangent Propagation". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 235–269. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_17. URL: https://doi.org/10.1007/978-3-642-35289-8_17.

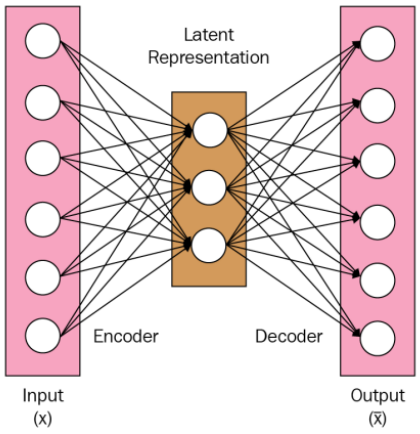

Figure 5: Illustration of an Auto-Encoder's architecture.

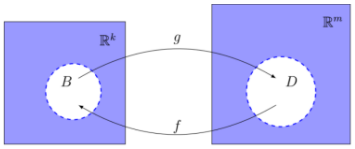

Figure 6: Mappings.

A MTC in-depth explanation

A.1 Contractive-Auto-Encoder (CAE) - the information extractor

Let's first understand what an Auto-encoder (AE) is:
First of all it is important to understand that it is a non-linear dimensionality reduction technique. It is an Artificial Neural Network and its architecture can be summarized in fig 5.

Now let's understand it:
Let's consider the dataset $\mathcal{D} = \{x_1, ..., x_n\}$. Examples $x_i \in R^m$ are i.i.d samples from an unknown distribution.

An AE It learns an *encoder* function h that maps an input $x \in R^d$ to a hidden representation $h(x) \in R^k$ with $k \leq m$. We have: $B \subset R^k$ and $D \subset R^m$.

The AE jointly learns a *decoder* function g that maps h back to the input space as $r = g(h(x))$ the reconstruction of x. See figure 6 for an intuitive understanding of these functions.
The encoder and decoder's parameters θ are learned by stochastic gradient descent to minimize the average reconstruction error $L(x, g(h(x))$ for the examples in the training set. The objective function of the AE is therefore:

$$\mathcal{J}_{AE}(\theta) = \sum_{x \in \mathcal{D}} L(x, g(h(x)))$$

Therefore an AutoEncoder is an interesting tool for reconstructing data from an unknown distribution.

It is expected that with respect to small changes of training data points, the autoencoder should encode very similar values. For instance in our problem we would like to have an encoder that encodes very similar values for a signal particle even if it is slightly rotated.

To fulfill this idea Contractive Auto-encoder (CAE) could be applied [1]. The CAE penalizes sensitivity of $f(x)$ to the inputs by $L2$ regularization term which is calculated as the squared Frobenius norm of the Jacobian for the encoder function f:

$$J(x) = \frac{\partial f}{\partial x}$$

The *cost function* of CAE is defined as:

$$\mathcal{J}_{CAE}(\theta) = \sum_{x \in \mathcal{D}} L(x, g(h(x))) + \lambda \left\| J(x) \right\|^2$$

Here θ is a set of the encoder and decoder parameters and λ is the non negative weight that controls the effect of penalizing the Jacobian's norm.

Now we have the desired Auto Encoder, one that encodes points to a similar value even if they have suffered small variances. The encoder's insensitivity to small variance is extremely interesting as we want to have a similar property in our neural network binary classifier.

Now we will study how to harness the information that the CAE extracts while learning how to reconstruct data even in the presence of variance. What interests us is the *information* that the encoder has learnt from the data that allows it to encode similar values for a data point x and a data point x' such that $x' = t(x)$, t being a transformation such as a translation or rotation.

A.2 Tangent vectors - information extracted

From [1] we learn that the encoder function gives us information about the variance of each datapoint. This information is encoded in the "chart" of each data point.

Let's first understand the vocabulary (see [2] for mathematical definitions):

Manifold: a space (or set) where we can locally (meaning close to a point, but not everywhere) assign a continuous mapping to the reals (in some dimension), with a continuous inverse. In our case we have a manifold that contains all the training data points

Tangent bundle of a manifold M: The tangent bundle of a differential manifold M is a collection of tangent planes at all points on the manifold M. Every tangent plane has its own chart which is defined as the tangent plane's coordinate system.

we extract a set of basis vectors for the local tangent space at each training point from the Contractive Auto-Encoder's learned parameters.

This is obtained with a Singular Value Decomposition (SVD) of the Jacobian of the encoder that maps each input to its learned representation:

Based on the hypothesis stated at the begining of [1], we then adopt the "generic prior" that class labels are likely to be insensitive to most directions within these local tangent spaces (ex: small translations, rotations or scalings usually do not change the particle's class).

A.3 CAE-Based tangent propagation - the classification algorithm

We can also leverage the extracted local charts when training a neural network. Following the tangent propagation approach of Simard et al. (1992), but exploiting our learned tangents, we encourage the output o of a neural network classifier to be insensitive to variations in the directions of the local chart of x by adding the following penalty to its supervised objective function:

B DANN in-depth explanation

B.1 Method description

The formal framework is the following: we consider classification tasks where X is the input space and Y = {0, 1,..., $L-1$ } is the set of L possible labels. Moreover, we have two different distributions over X×Y , called the source domain D_s and the target domain D_t. An unsupervised domain adaptation learning algorithm is then provided with a labeled source sample S drawn i.i.d. from D_s and an unlabeled target sample T drawn i.i.d. from D_t^X, where D_t^X is the marginal distribution of D_t over X,

$$S = \{\mathbf{x}_i, y_i\}_{i=1}^{n} \sim (D_s)^n \ ; \ T = \{\mathbf{x}_i\}_{i=n+1}^{N} \sim (D_t^X)^{n'}$$

with $N = n + n'$ being the total number of samples. The goal of the learning algorithm is to build a classifier $\eta : X \longrightarrow Y$ with a low target risk

$$R_{D_t}(\eta) = Pr_{(\mathbf{x},y)\sim(D_t)} \left(\eta(\mathbf{x}) \neq y\right)$$

while having no information about the labels of D_t.

To describe the Domain Adversarial Neural Network (DANN) architechture, we define the following formulation. Let $G_f\left(\cdot;\theta_f\right)$ be the D-dimensional neural network feature extractor, with parameters θ_f. Also, let $G_y\left(\cdot;\theta_y\right)$ be the part of DANN that computes the network's label prediction output, with parameters θ_y, while $G_d\left(\cdot;\theta_d\right)$ finally corresponds to the computation of the domain prediction output of the network, with parameters θ_d. We will note the prediction loss and the domain loss respectively by:

$$\mathcal{L}_y^i(\theta_f, \theta_y) = \mathcal{L}_y(G_y(G_f(\mathbf{x}_i;\theta_f);\theta_y), y_i) \tag{1}$$

$$\mathcal{L}_d^i(\theta_f, \theta_d) = \mathcal{L}_d(G_d(G_f(\mathbf{x}_i;\theta_f);\theta_d), d_i) \tag{2}$$

where $\mathbf{x}_i$ is a single datapoint, y_i is the corresponding class and d_i is the corresponding domain.
Then, training the DANN consists in optimizing

$$E(\theta_f, \theta_y, \theta_d) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_y^i(\theta_f, \theta_y) \; - \lambda \left(\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_d^i(\theta_f, \theta_d) + \frac{1}{n'}\sum_{i=n+1}^{N} \mathcal{L}_d^i(\theta_f, \theta_d) \right) \quad (3)$$

by finding the saddle point $\hat{\theta_f}, \hat{\theta_y}, \hat{\theta_d}$ such that

$$(\hat{\theta_f}, \hat{\theta_y}) = \arg\min_{\theta_f, \theta_y} E(\theta_f, \theta_y, \hat{\theta_d}) \quad (4)$$

$$\hat{\theta_d} = \arg\max_{\theta_d} E(\hat{\theta_f}, \hat{\theta_y}, \theta_d) \quad (5)$$

A saddle point as defined can be found as a stationary point of the following gradient updates:

$$\theta_f \longleftarrow \theta_f \; - \mu \left(\frac{\partial \mathcal{L}_y^i}{\partial \theta_f} \; - \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_f} \right) \quad (6)$$

$$\theta_y \longleftarrow \theta_y \; - \mu \frac{\partial \mathcal{L}_y^i}{\partial \theta_y} \quad (7)$$

$$\theta_d \longleftarrow \theta_d \; - \mu\lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_d} \quad (8)$$

where μ is the learning rate. We use stochastic estimates of these gradients, by sampling examples from the data set.
The updates of Equations (6-8) are very similar to stochastic gradient descent (SGD) updates for a feed-forward deep model that comprises feature extractor fed into the label predictor and into the domain classifier (with loss weighted by λ). The important aspect to notice is in Equation (6): the gradients from the class and domain predictors are subtracted, instead of being summed, which is an important difference as otherwise it would try to make features dissimilar across domains in order to minimize the domain classification loss. Such a reduction can be accomplished by introducing a special gradient reversal layer (GRL), that has no parameters associated with it: during the forward propagation, the GRL acts as an identity transformation, while during the backpropagation however, the GRL takes the gradient from the subsequent level and changes its sign, i.e., multiplies it by -1, before passing it to the preceding layer. The layer requires no parameter update. The GRL as defined above is inserted between the feature extractor G_f and the domain classifier G_d, resulting in the architecture depicted in Figure 7. As the backpropagation process passes through the GRL, the partial derivatives of the loss that is downstream the GRL (i.e., $\mathcal{L}_d$) w.r.t. the layer parameters that are upstream the GRL (i.e., θ_f) get multiplied by -1. Therefore, running SGD in the resulting model implements the updates of Equations (6-8) and converges to a saddle point of Equation (3).

The proposed DANN architecture is more intuitive by looking at Figure 7. It indeed includes a deep feature extractor (green) and a deep label predictor (blue), which together form a standard feed-forward architecture. Unsupervised domain adaptation is achieved by adding a domain classifier (red) connected to the feature extractor via a gradient reversal layer that multiplies the gradient by a certain negative constant during the back-propagation-based training. Otherwise, the training proceeds standardly and minimizes the label prediction loss (for source examples) and the domain classification loss (for all samples). Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the domain-invariant features.

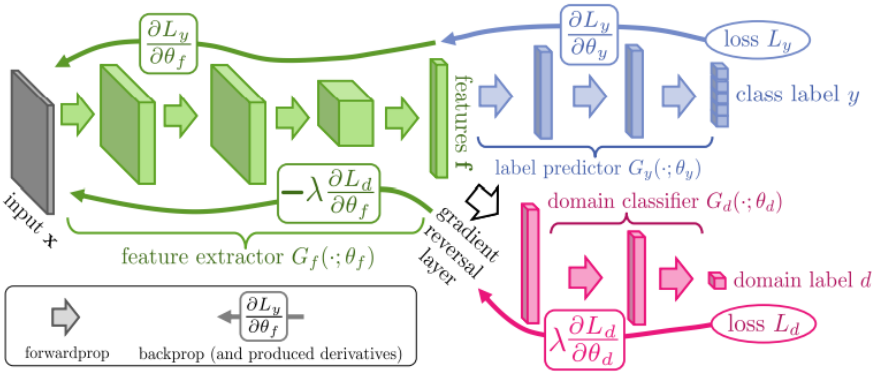

Figure 7: Representation of the Domain Adversarial Neural Network