Domain Adversarial Neural Networks

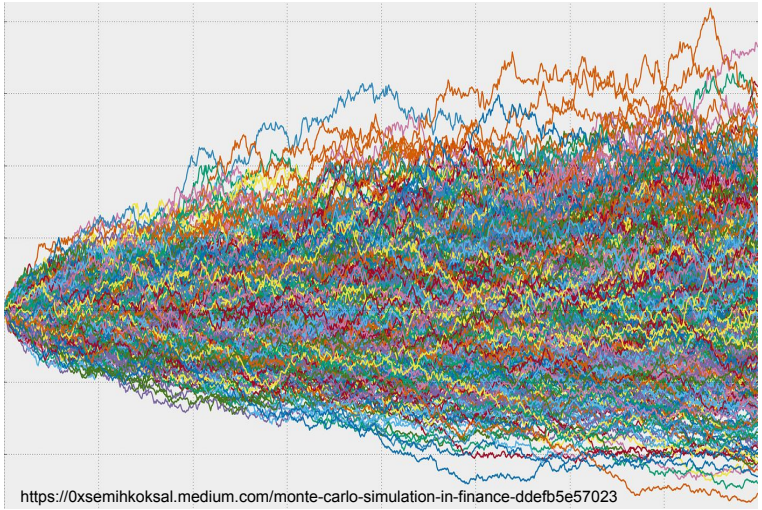
A framework to train on labeled (simulated) and unlabeled (real) data

Marcus Engsig - bgr911
Gor Nahapetyan - xzn779
Frederik Hansen - bcp695
Johan Sieborg - hnw224

Table of Contents

Problem Statement

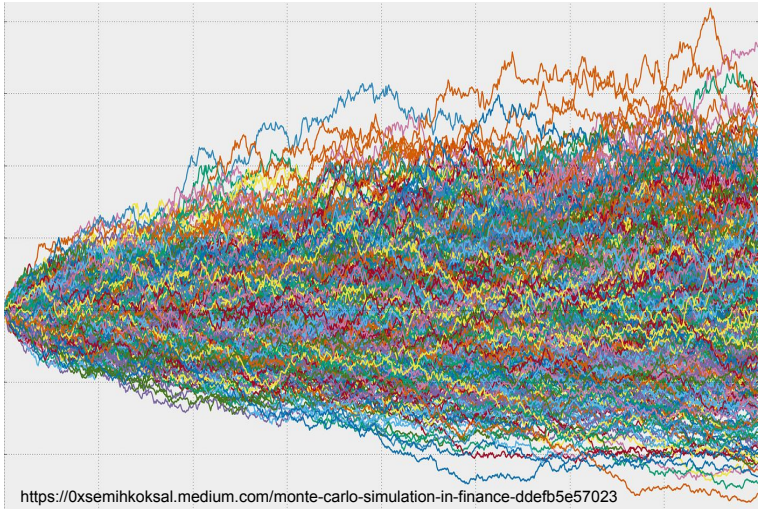
Frequently, **simulated data** is **easy** to obtain.



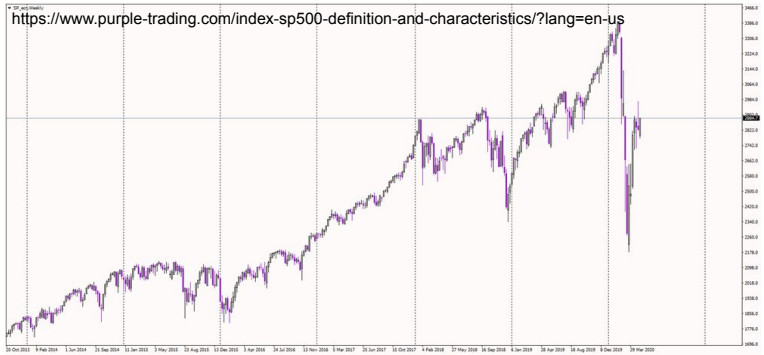
Problem Statement

Frequently, **simulated data** is **easy** to obtain.

Conversely, **real data** is generally **difficult** and **expensive** to gather.



https://0xsemihkoksal.medium.com/monte-carlo-simulation-in-finance-ddefb5e57023



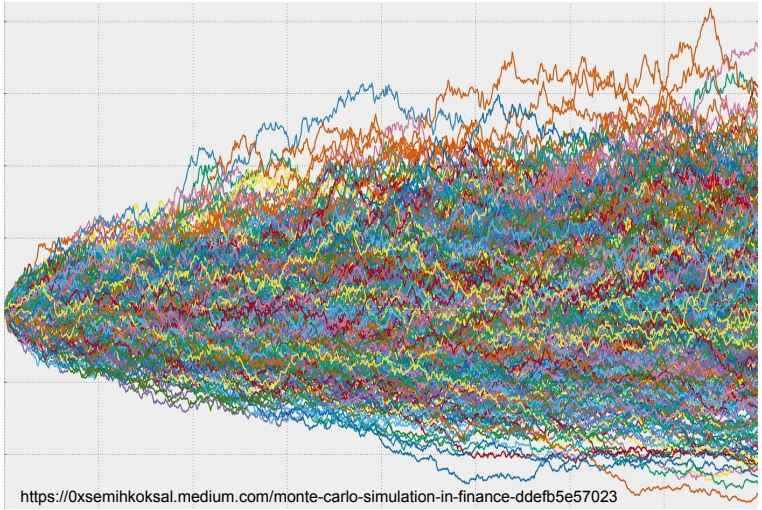
https://www.purple-trading.com/index-sp500-definition-and-characteristics/?lang=en-us

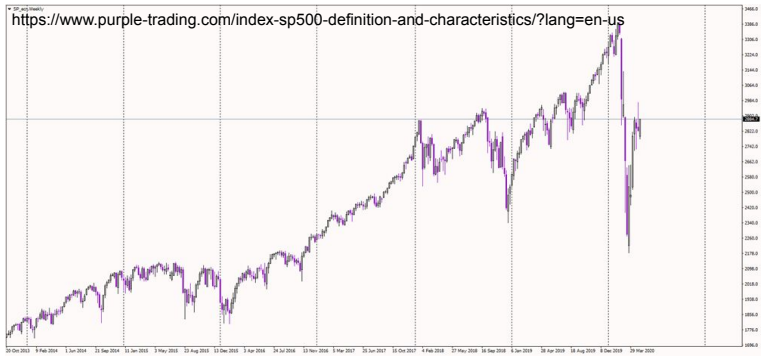
Problem Statement

Frequently, **simulated data** is **easy** to obtain.

Conversely, **real data** is generally **difficult** and **expensive** to gather.

On top of that, **real data** tends to be **difficult** and sometimes impossible to **label**.





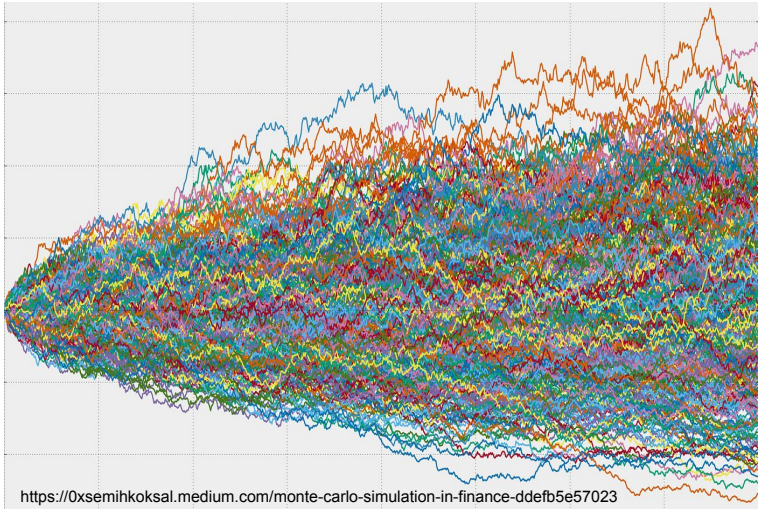
Problem Statement

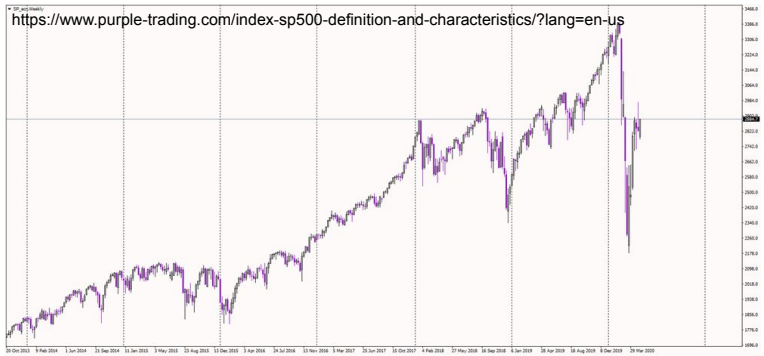
Frequently, **simulated data** is **easy** to obtain.

Conversely, **real data** is generally **difficult** and **expensive** to gather.

On top of that, **real data** tends to be **difficult** and sometimes impossible to **label**.

How can we **train** on **simulated data**, and **generalize** our model to **real data** without overfitting (to the simulated data).





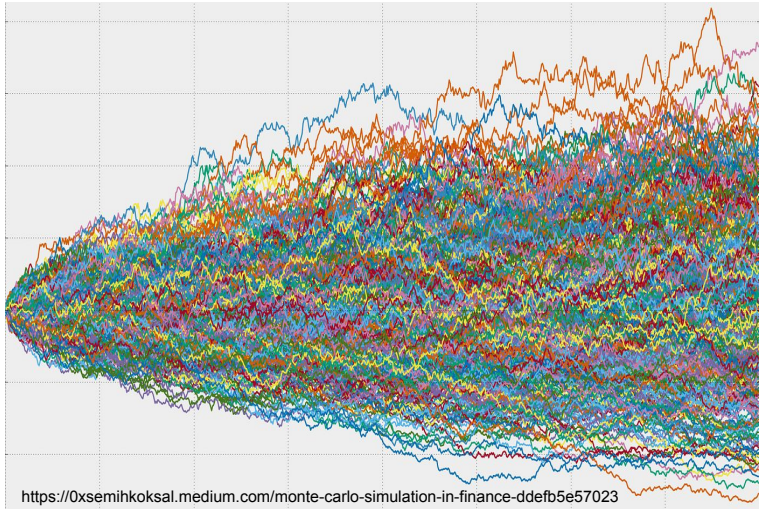
Problem Statement

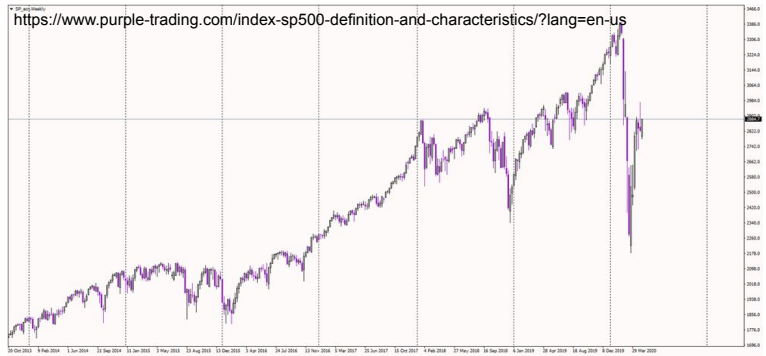
Frequently, **simulated data** is **easy** to obtain.

Conversely, **real data** is generally **difficult** and **expensive** to gather.

On top of that, **real data** tends to be **difficult** and sometimes impossible to **label**.

How can we **train** on **simulated data**, and **generalize** our model to **real data** without overfitting (to the simulated data).





Dataset Archetype

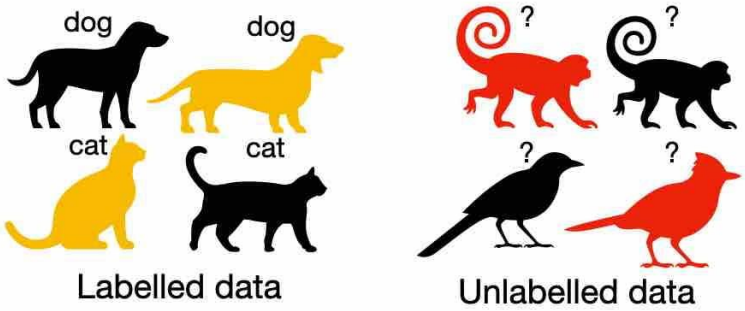
Our dataset consists of:

1. a set of **labelled** ‘simulated’ data.
2. a set of **unlabelled** ‘real’ data.

Dataset Archetype

Our dataset consists of:

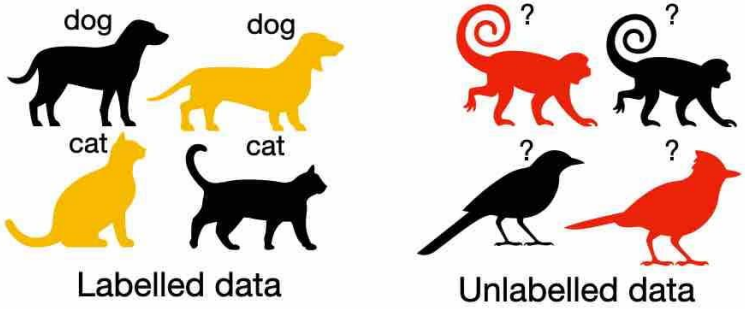
1. a set of **labelled** ‘simulated’ data.
2. a set of **unlabelled** ‘real’ data.

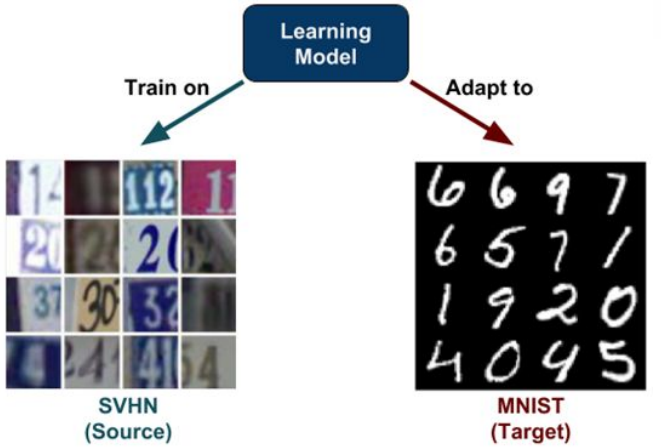


Dataset Archetype

Our dataset consists of:

1. a set of **labelled** ‘simulated’ data.
2. a set of **unlabelled** ‘real’ data.



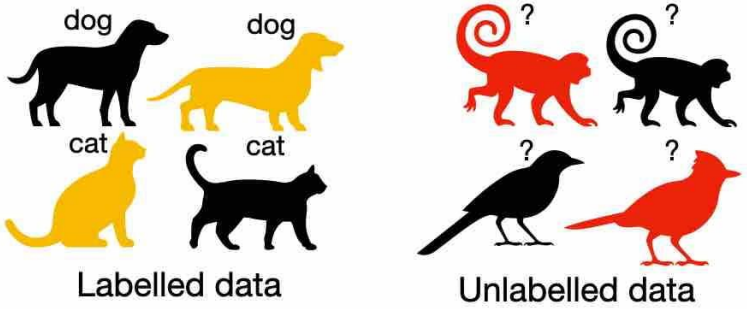


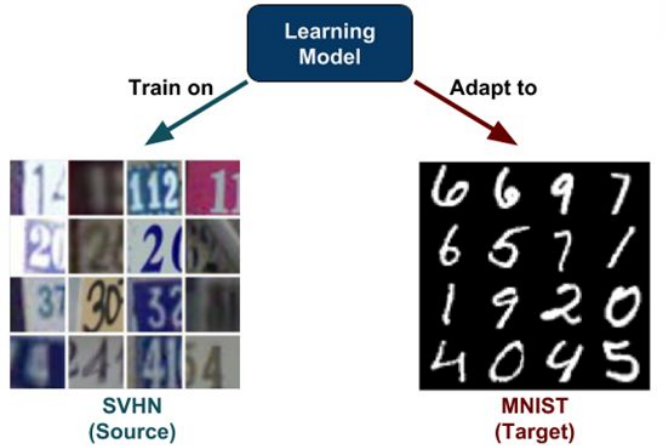
Dataset Archetype

Our dataset consists of:

1. a set of **labelled** ‘simulated’ data.
2. a set of **unlabelled** ‘real’ data.

Even though the ‘real’ data is **unlabelled**, there is still some hidden and **mineable information.**





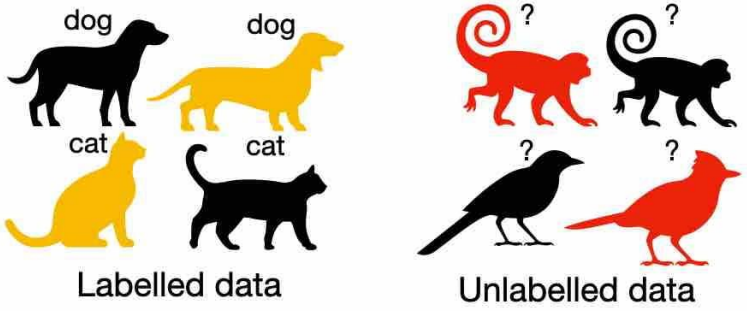
Dataset Archetype

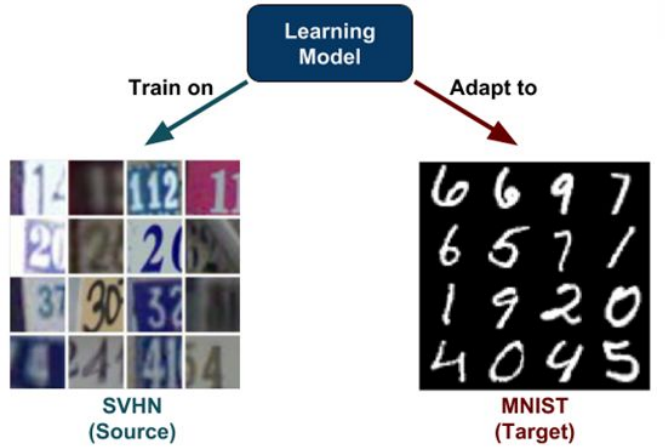
Our dataset consists of:

1. a set of **labelled** ‘simulated’ data.
2. a set of **unlabelled** ‘real’ data.

Even though the ‘real’ data is **unlabelled**, there is still some hidden and **mineable information**.

The label of being ‘simulated’ vs. ‘real’.



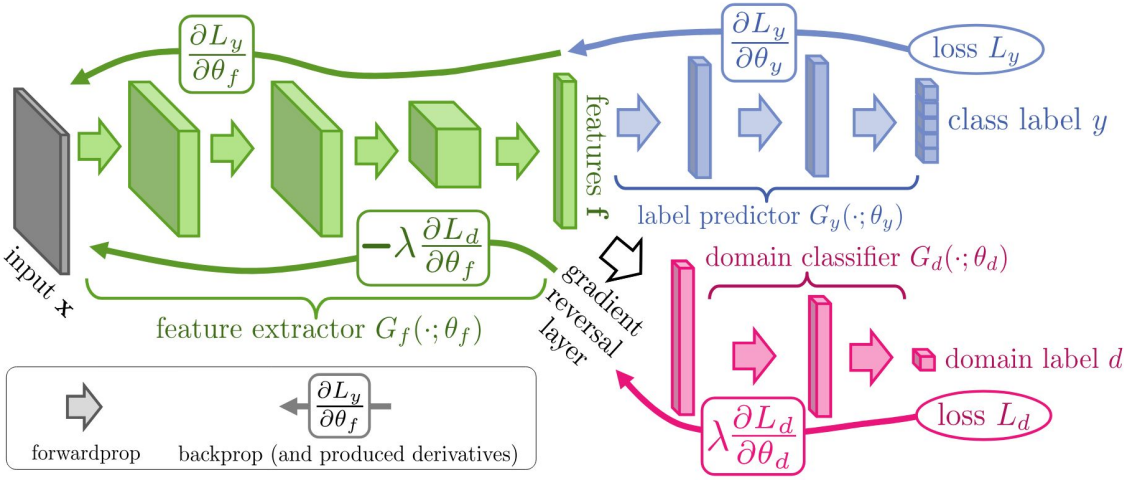


Domain Adversarial Neural Networks.

How can we use the ‘real’ vs. ‘simulated’ **label** to **generalize** our model to the ‘real’ data?

Domain Adversarial Neural Networks.

How can we use the ‘real’ vs. ‘simulated’ **label** to **generalize** our model to the ‘real’ data?

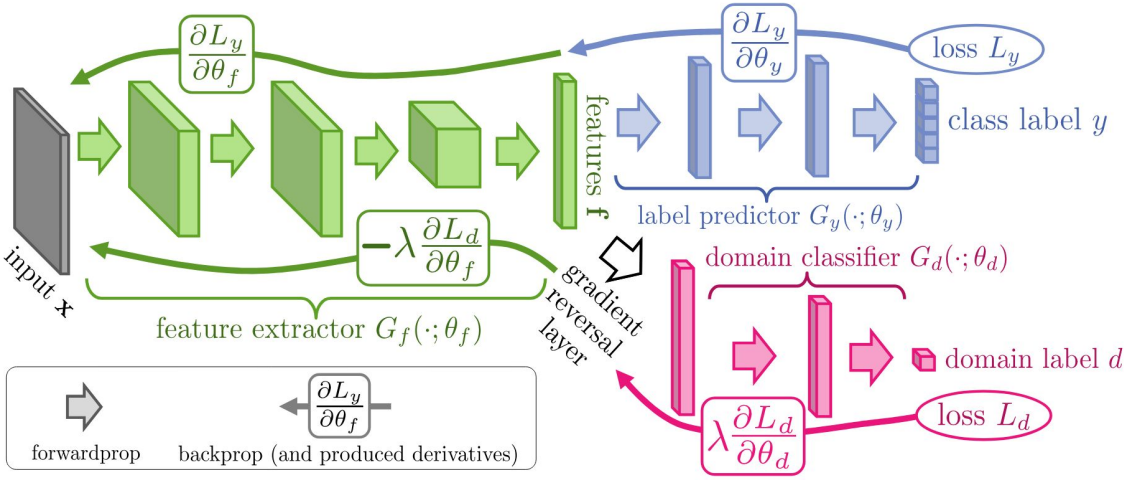


Y. Ganin et. al., 2016.

The short answer… **Adversarial Neural Nets**.

Domain Adversarial Neural Networks.

How can we use the ‘real’ vs. ‘simulated’ **label** to **generalize** our model to the ‘real’ data?



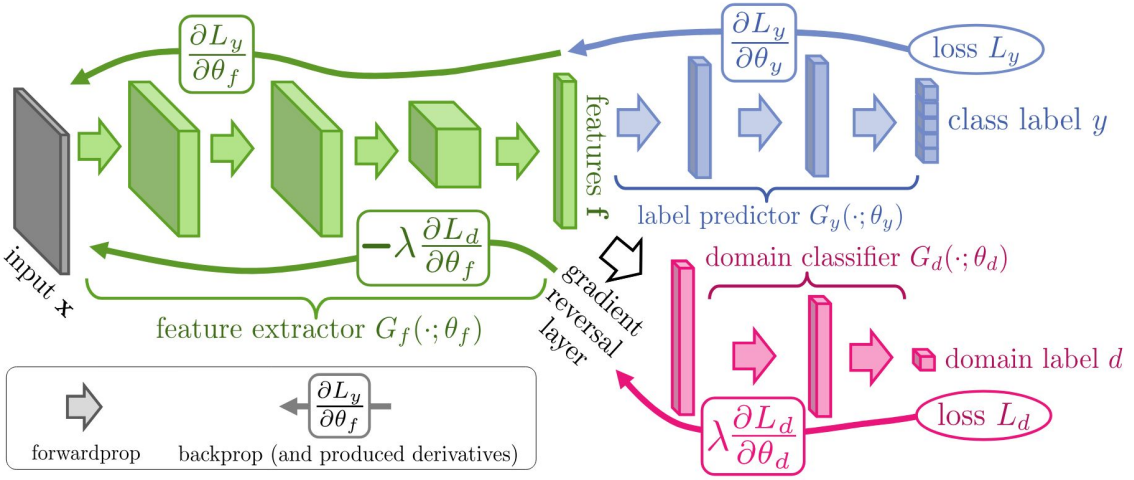
Y. Ganin et. al., 2016.

The short answer… **Adversarial Neural Nets**.

1. The feature extractor tries to **mine relevant information** from the real or simulated data that is **indistinguishable**, despite the **origin** of the data (think auto-encoder).

Domain Adversarial Neural Networks.

How can we use the ‘real’ vs. ‘simulated’ **label** to **generalize** our model to the ‘real’ data?



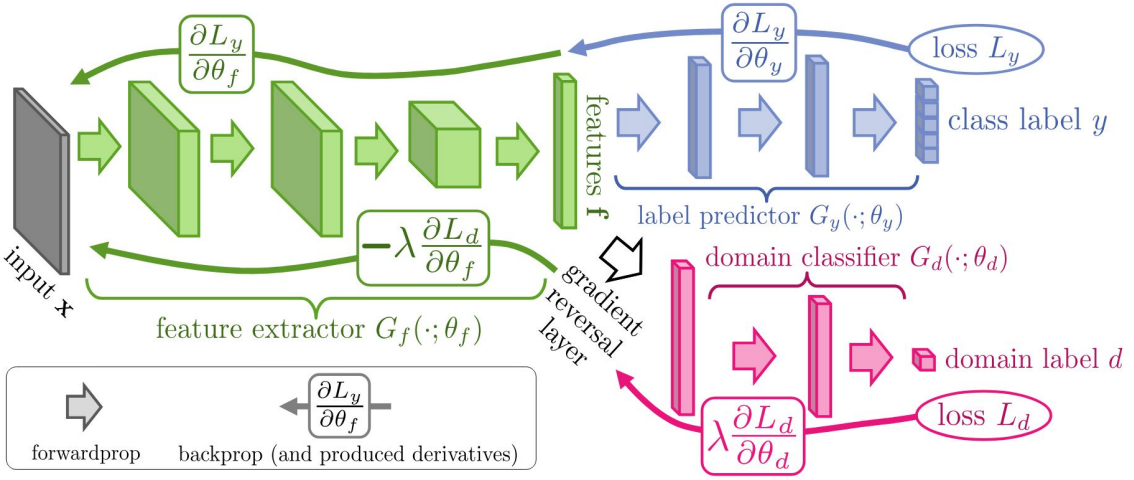
Y. Ganin et. al., 2016.

The short answer… **Adversarial Neural Nets**.

1. The feature extractor tries to **mine relevant information** from the real or simulated data that is **indistinguishable**, despite the **origin** of the data (think auto-encoder).
2. The discriminator (domain classifier) tries to **classify** whether the data is **real** or **simulated**.

Domain Adversarial Neural Networks.

How can we use the ‘real’ vs. ‘simulated’ **label** to **generalize** our model to the ‘real’ data?



Y. Ganin et. al., 2016.

The short answer… **Adversarial Neural Nets**.

1. The feature extractor tries to **mine relevant information** from the real or simulated data that is **indistinguishable**, despite the **origin** of the data (think auto-encoder).
2. The discriminator (domain classifier) tries to **classify** whether the data is **real** or **simulated**.
3. The label predictor attempts to **classify** (main task) based on the **main-objective** labels from **only** the **simulated** data.

Loss Function

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

Loss Function

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

Classifier Loss

Loss Function

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

Classifier Loss

Discriminator Loss

Loss Function

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

Classifier Loss

Discriminator Loss

Where the loss function is Binary Cross Entropy (BCE)

$$\mathcal{L}_i = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

Loss Function

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

Classifier Loss

Discriminator Loss

Where the loss function is Binary Cross Entropy (BCE)

$$\mathcal{L}_i = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

How do we implement this?...

Loss Function

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

Classifier Loss

Discriminator Loss

How do we implement this?...

Loss Function

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

How do we implement this?...

Loss Function

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

Gradient Reversal

Loss Function

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

Gradient Reversal

The job of the **Gradient Reversal Layer** (GRL):

Loss Function

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

Gradient Reversal

The job of the **Gradient Reversal Layer** (GRL):

1. To **reverse** the **gradient** from the discriminator (the minus sign).

Loss Function

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

Gradient Reversal

The job of the **Gradient Reversal Layer** (GRL):

1. To **reverse** the **gradient** from the discriminator (the minus sign).
2. To **scale** the **gradient** by a factor **λ**

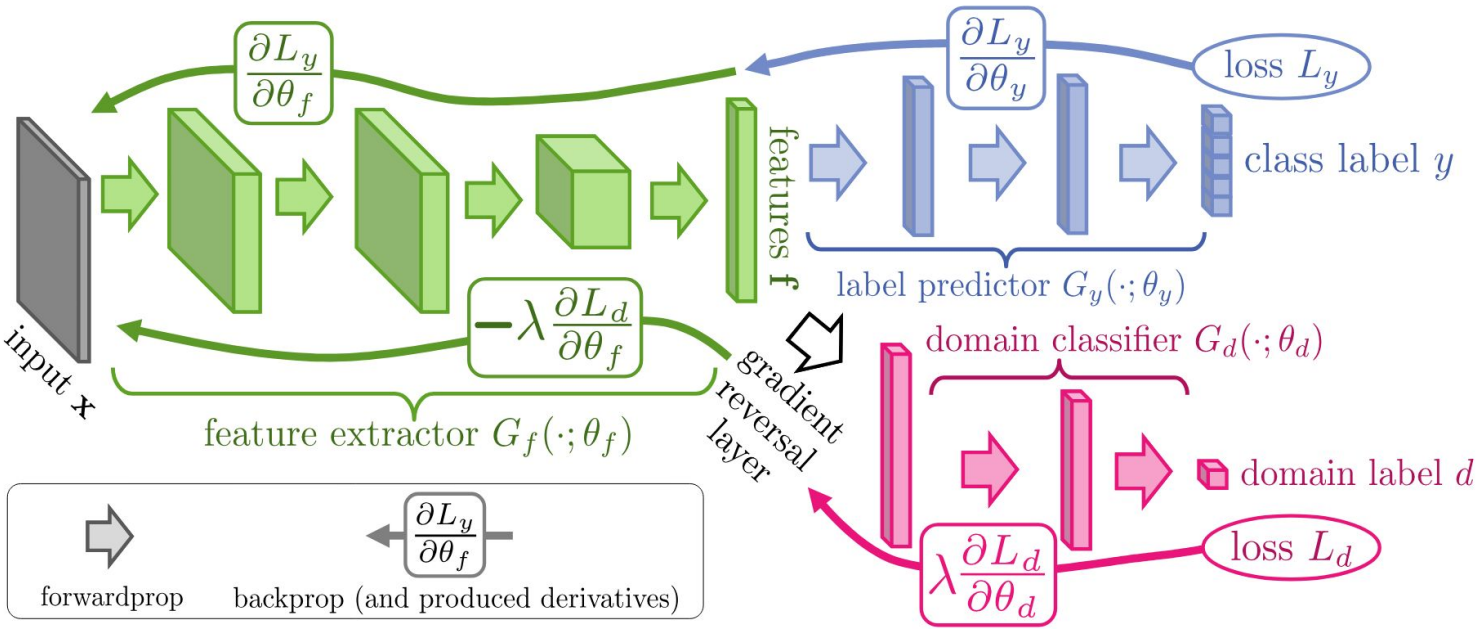
Loss Function

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

Gradient Reversal

The job of the **Gradient Reversal Layer** (GRL):

1. To **reverse** the **gradient** from the domain classifier (the minus sign).
2. To **scale** the **gradient** by a factor **λ**
3. Train the Feature Extractor to '**fool**' the domain classifier → **reduce** simulated and real data into an **indistinguishable latent space**.



Y. Ganin et. al., 2016.

An important anecdote

Y. Ganin et. al., showed that DANNs perform **best** where the is a **domain shift** between the ‘real’ and simulated data.

Domain-Adversarial Training of Neural Networks

Yaroslav Ganin GANIN@SKOLTECH.RU
Evgeniya Ustinova EVGENIYA.USTINOVA@SKOLTECH.RU
Skolkovo Institute of Science and Technology (Skoltech)
Skolkovo, Moscow Region, Russia

Hana Ajakan HANA.AJAKAN.1@ULAVAL.CA
Pascal Germain PASCAL.GERMAIN@IFT.ULAVAL.CA
Département d'informatique et de génie logiciel, Université Laval
Québec, Canada, G1V 0A6

Hugo Larochelle HUGO.LAROCHELLE@USHERBROOKE.CA
Département d'informatique, Université de Sherbrooke
Québec, Canada, J1K 2R1

François Laviolette FRANCOIS.LAVIOLETTE@IFT.ULAVAL.CA
Mario Marchand MARIO.MARCHAND@IFT.ULAVAL.CA
Département d'informatique et de génie logiciel, Université Laval
Québec, Canada, G1V 0A6

Victor Lempitsky LEMPITSKY@SKOLTECH.RU
Skolkovo Institute of Science and Technology (Skoltech)
Skolkovo, Moscow Region, Russia

Editor: Urun Dogan, Marius Kloft, Francesco Orabona, and Tatiana Tommasi

Abstract

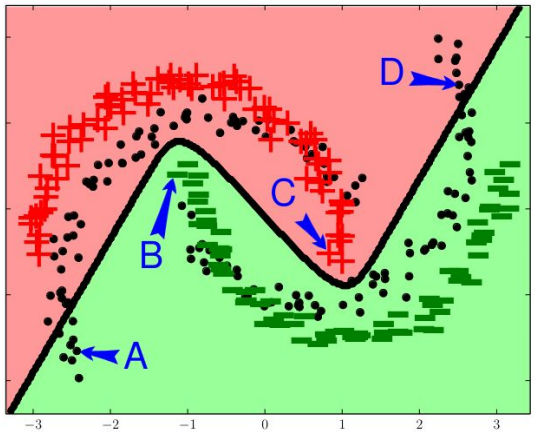
We introduce a new representation learning approach for domain adaptation, in which data at training and test time come from similar but different distributions. Our approach is directly inspired by the theory on domain adaptation suggesting that, for effective domain transfer to be achieved, predictions must be made based on features that cannot discriminate between the training (source) and test (target) domains.

The approach implements this idea in the context of neural network architectures that are trained on labeled data from the source domain and unlabeled data from the target domain (no labeled target-domain data is necessary). 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. We show that this adaptation behaviour can be achieved in almost any feed-forward model by augmenting it with few standard layers and a new *gradient reversal* layer. The resulting augmented architecture can be trained using standard backpropagation and stochastic gradient descent, and can thus be implemented with little effort using any of the deep learning packages.

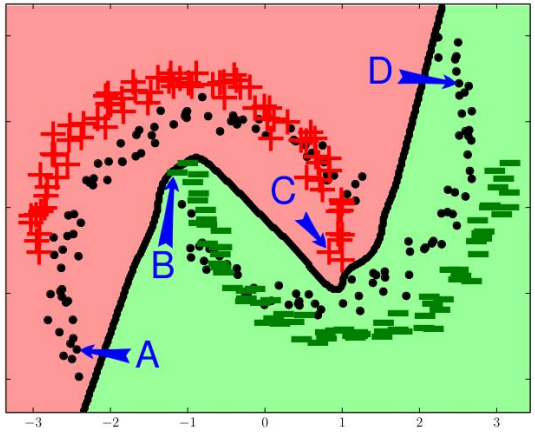
An important anecdote

Y. Ganin et. al., showed that DANNs perform **best** where the is a **domain shift** between the ‘real’ and simulated data.

MLP



Adversarial

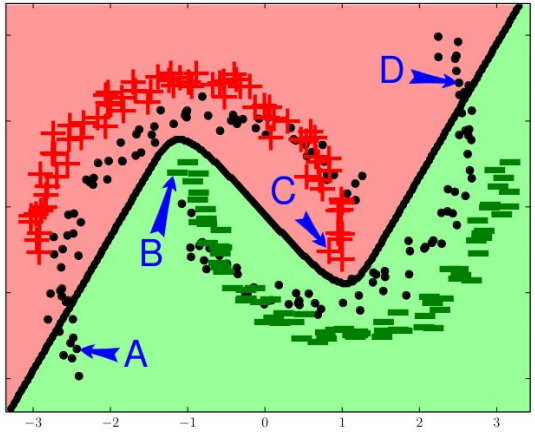


(Ganin et al. 2015)

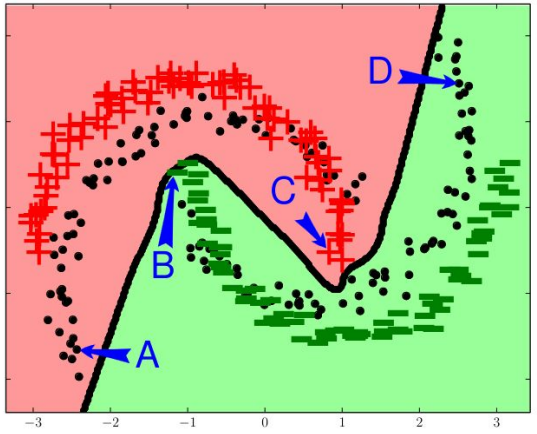
An important anecdote

Y. Ganin et. al., showed that DANNs perform **best** where the is a **domain shift** between the ‘real’ and simulated data.

MLP

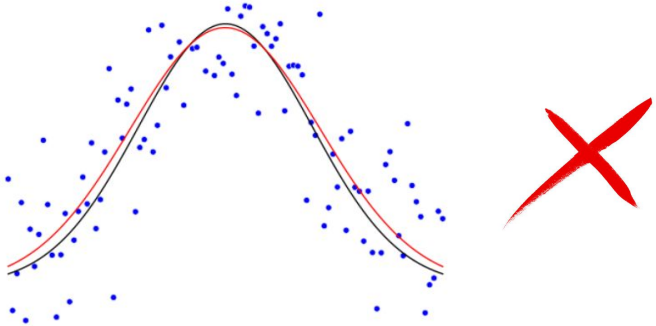


Adversarial



The data must be **fundamentally distinguishable**.

Random noise augmentation:



Our Data Augmentation

We have Monte-Carlo **ALEPH** data.

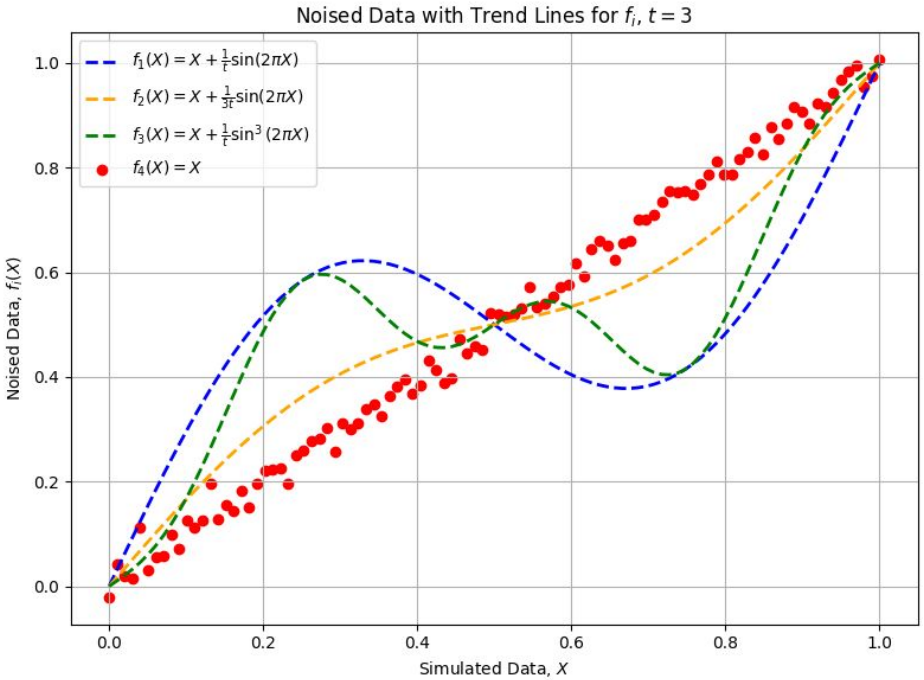
1. 6 parameter input
2. 1 parameter classification output

Our Data Augmentation

We have Monte-Carlo **ALEPH** data.

1. 6 parameter input
2. 1 parameter classification output

We augment our data X with a ‘noise’ parameter **t** with four functions f_i.

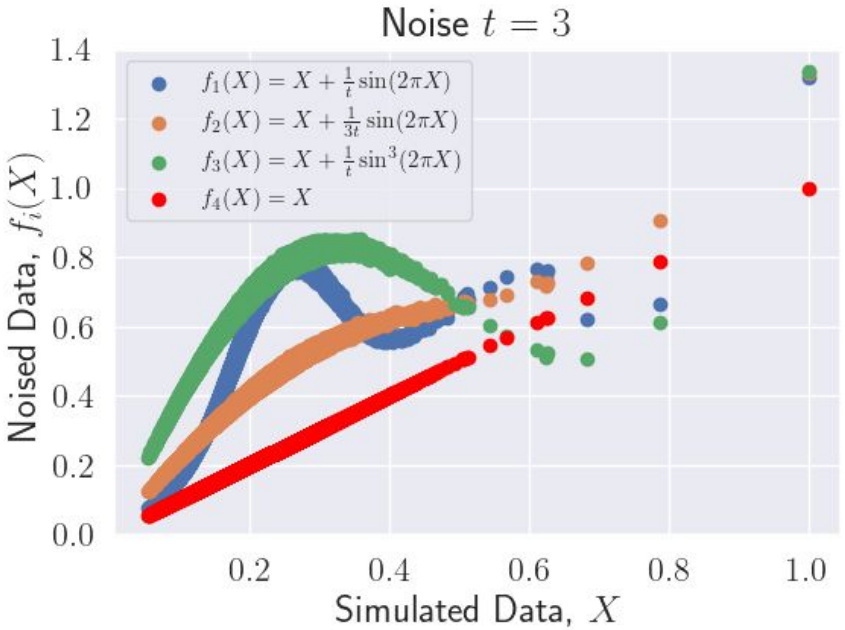


Our Data Augmentation

We have Monte-Carlo **ALEPH** data.

1. 6 parameter input
2. 1 parameter classification output

We augment our data X with a 'noise' parameter **t** with four functions f_i.



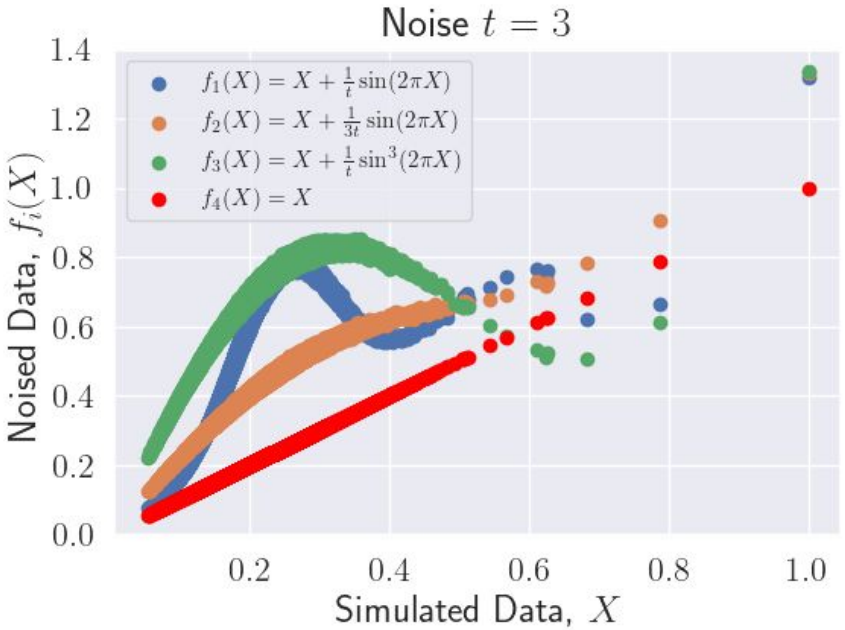
Our Data Augmentation

We have Monte-Carlo **ALEPH** data.

1. 6 parameter input
2. 1 parameter classification output

We augment our data X with a 'noise' parameter **t** with four functions f_i.

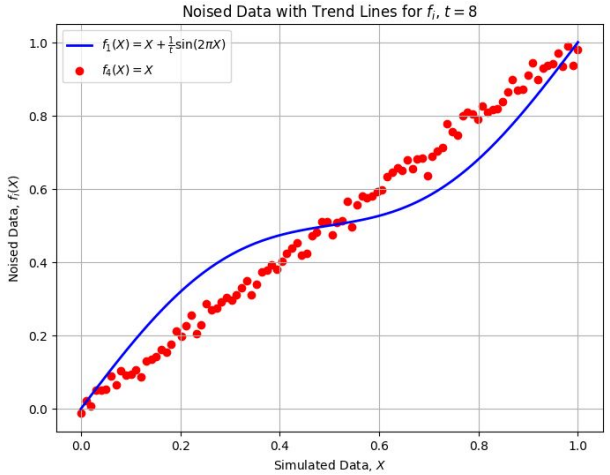
Transformed data has **no truth values** during training.



MLP - Our Null Hypothesis

We train an MLP: **t = 8**

1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

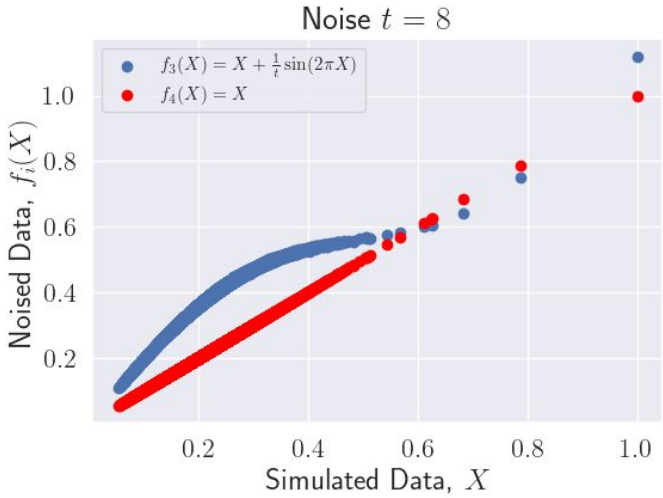


MLP - Our Null Hypothesis

We train an MLP: **t = 8**

1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

We **train** it on **‘simulated’ data**, and use it to **predict** the **‘real’ data**.

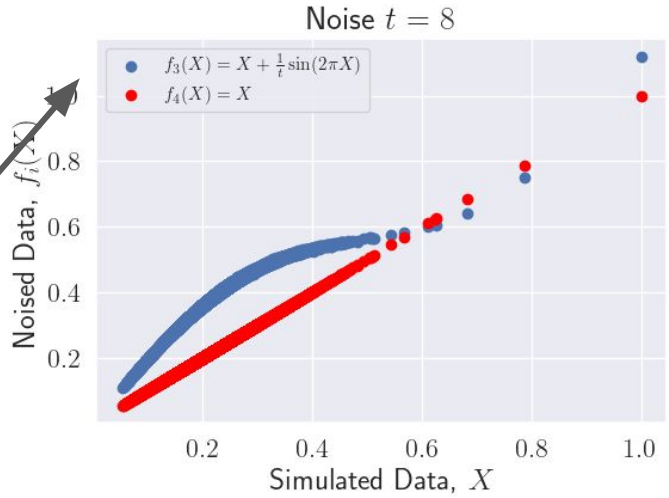


MLP - Our Null Hypothesis

We train an MLP: **t = 8**

1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

We **train** it on **'simulated' data**, and use it to **predict** the **'real' data**.

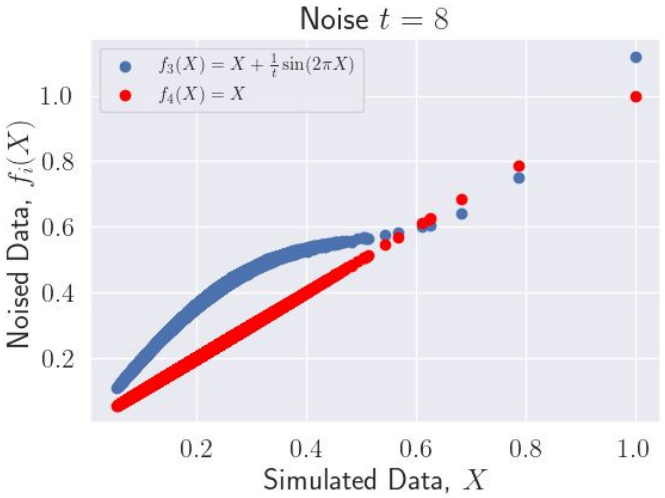


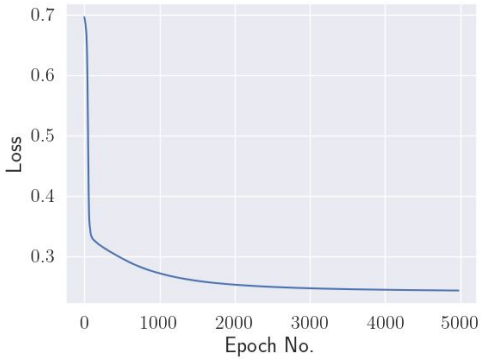
MLP - Our Null Hypothesis

We train an MLP: **t = 8**

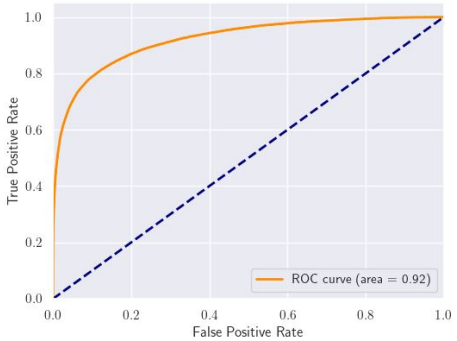
1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

We **train** it on **'simulated' data**, and use it to **predict** the **'real' data**.





Real Accuracy = 0.889



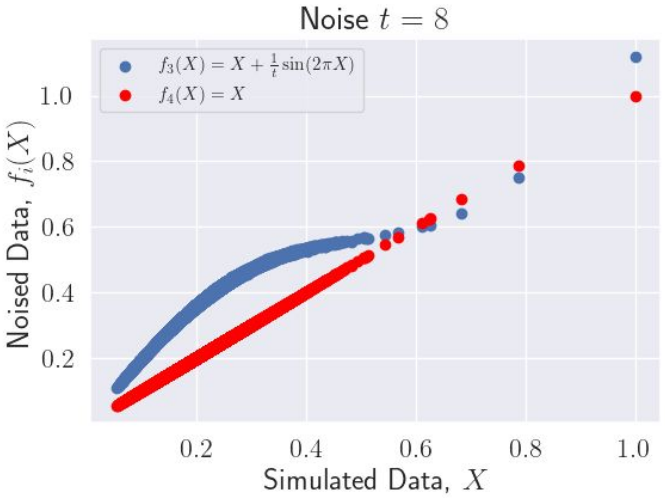
MLP - Our Null Hypothesis

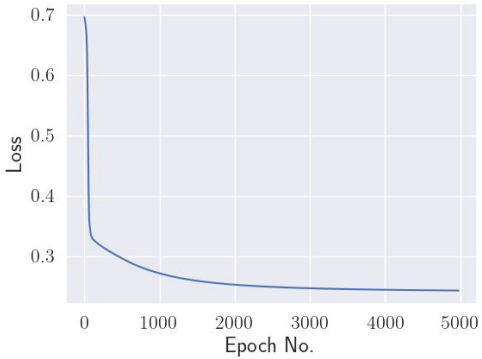
We train an MLP: **t = 8**

1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

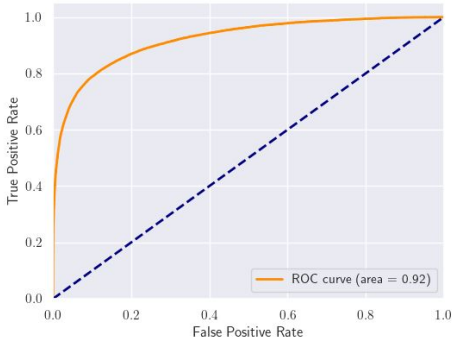
We **train** it on **'simulated' data**, and use it to **predict** the **'real' data**.

For small transformations MLPs do great!





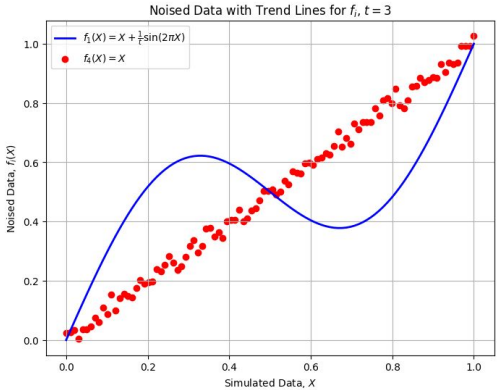
Real Accuracy = 0.889



MLP - Our Null Hypothesis… Lets increase the noise!

We train an MLP: **t = 3**

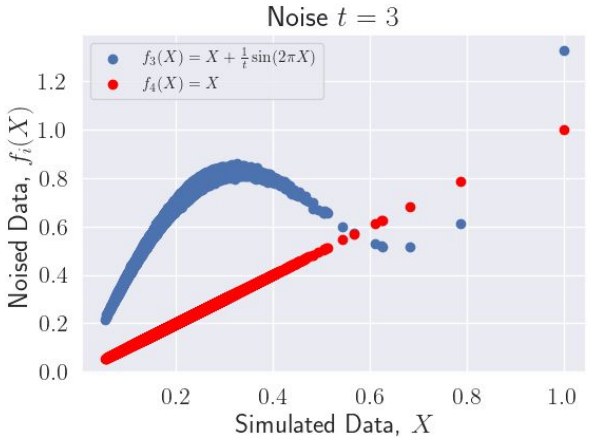
1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.



MLP - Our Null Hypothesis… Lets increase the noise!

We train an MLP: **t = 3**

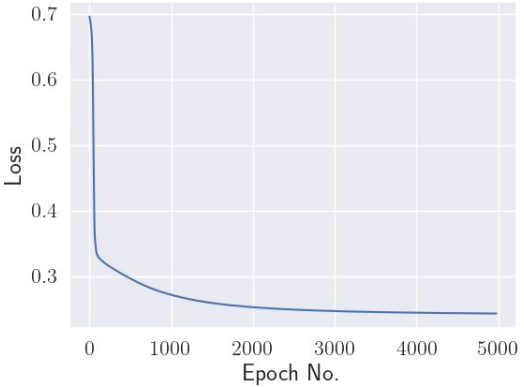
1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

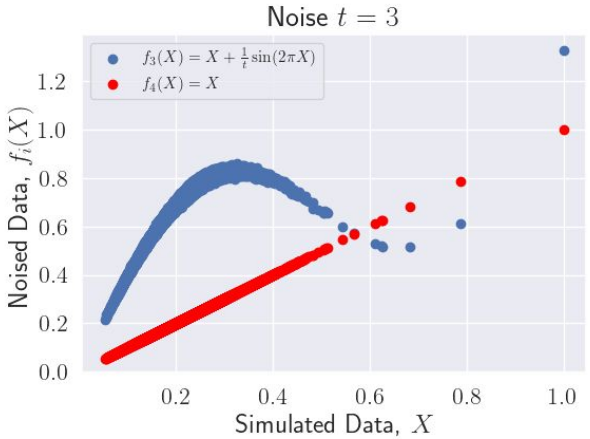


MLP - Our Null Hypothesis… Lets increase the noise!

We train an MLP: **t = 3**

1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

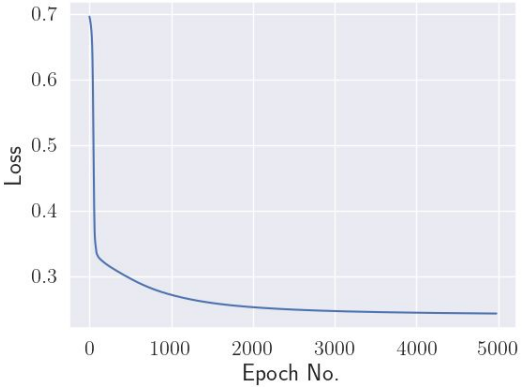


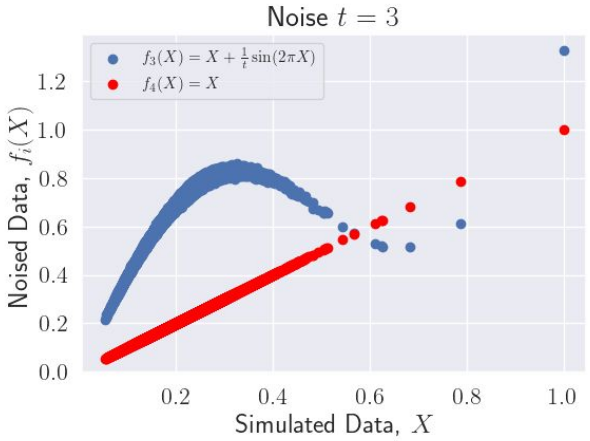


MLP - Our Null Hypothesis… Lets increase the noise!

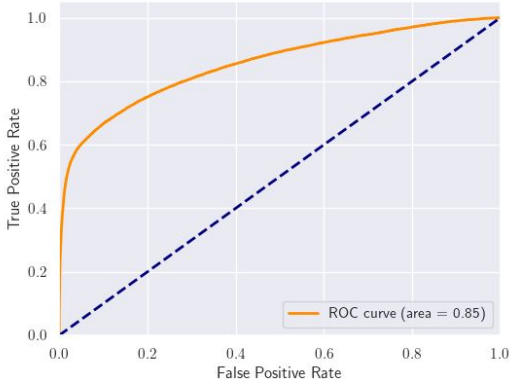
We train an MLP: **t = 3**

1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.





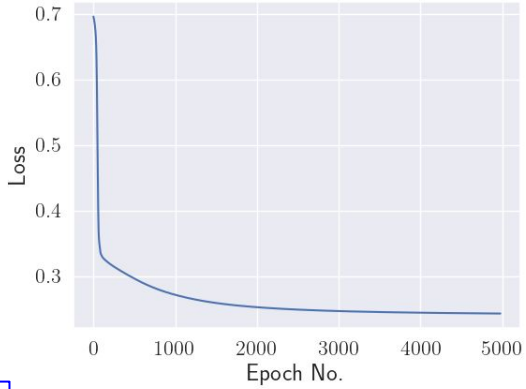
Real Accuracy = 0.693

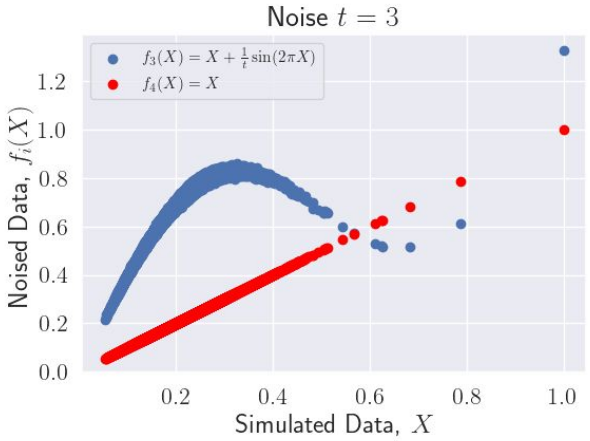


MLP - Our Null Hypothesis… Lets increase the noise!

We train an MLP: **t = 3**

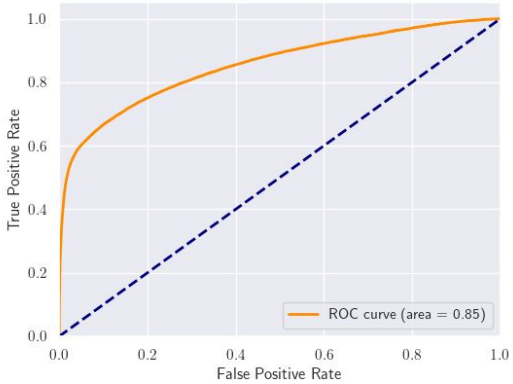
1. Architecture: 6x10x10x10x5x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.



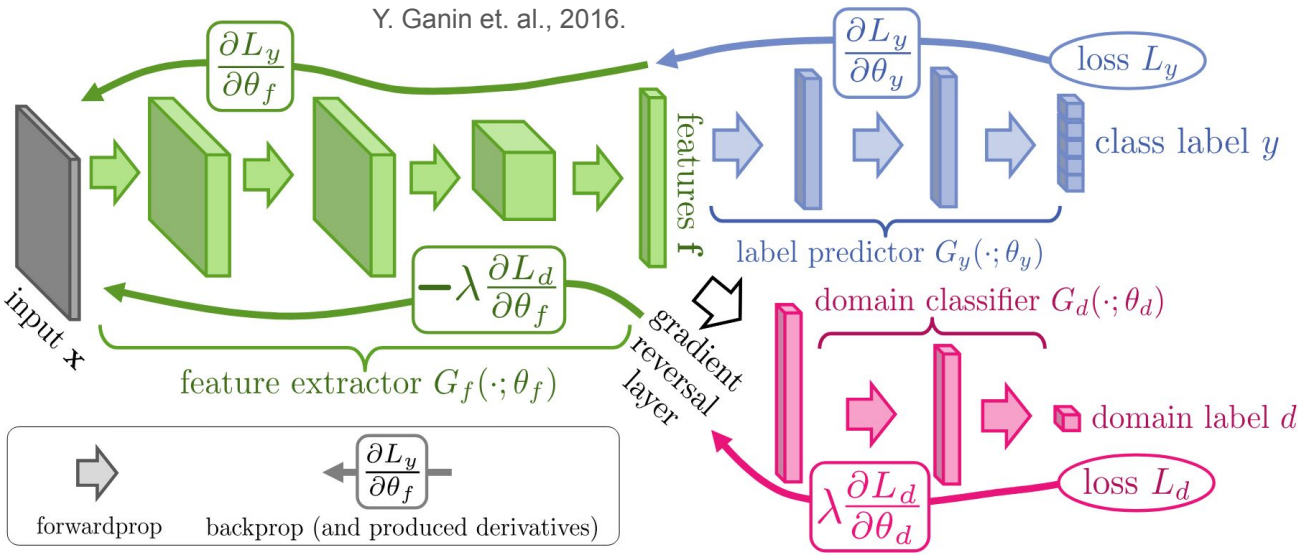


The MLP **breaks** when there is a clear **domain shift**.

Real Accuracy = 0.693



A quick reminder on DANNs



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

Domain Adversarial Neural Networks on noisy data

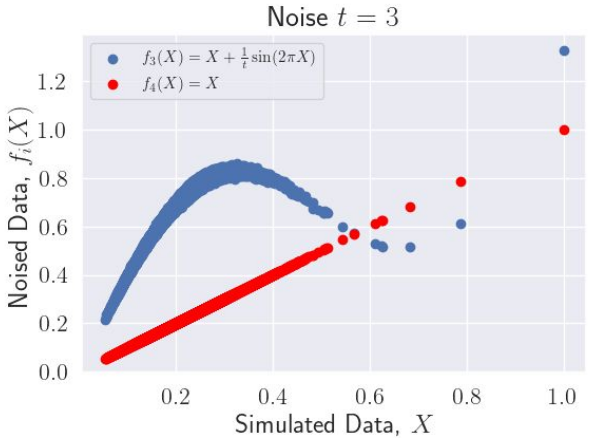
We train an DANN: **t = 3, λ = 50**

1. Architecture: Main task 6x10x10x10x5x1,
   Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

Domain Adversarial Neural Networks on noisy data

We train an DANN: **t = 3, λ = 50**

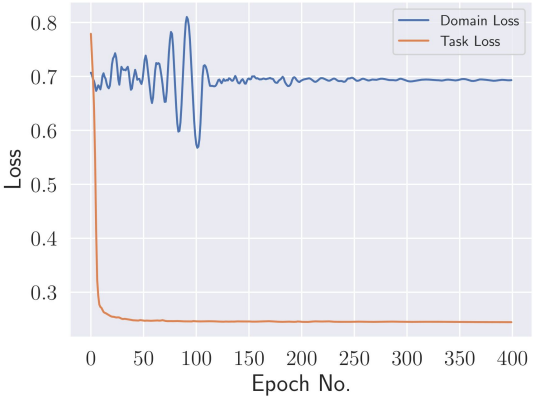
1. Architecture: Main task 6x10x10x10x5x1,
   Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

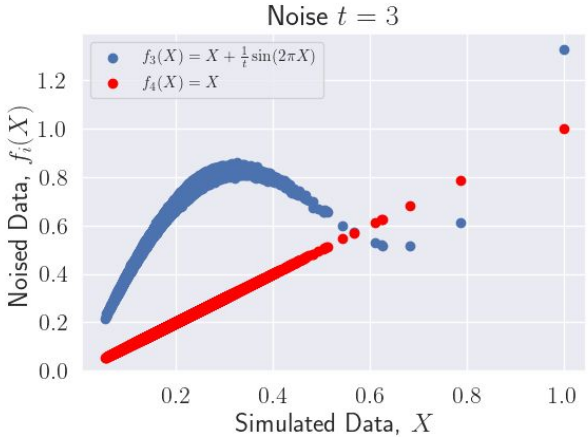


Domain Adversarial Neural Networks on noisy data

We train an DANN: **t = 3, λ = 50**

1. Architecture: Main task 6x10x10x10x5x1, Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

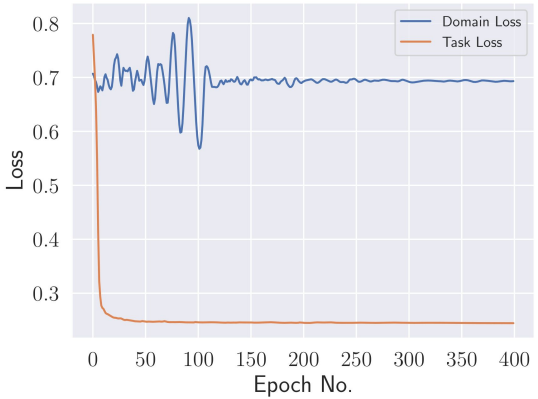


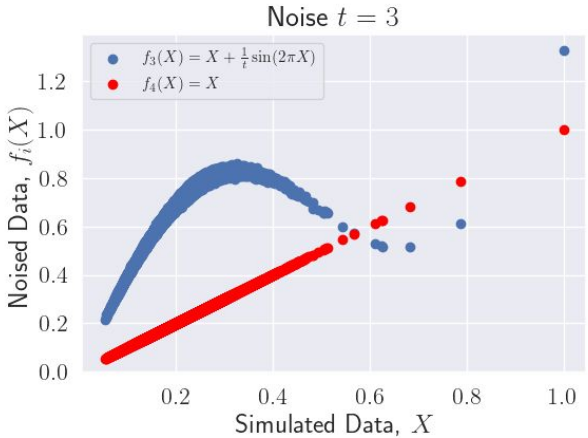


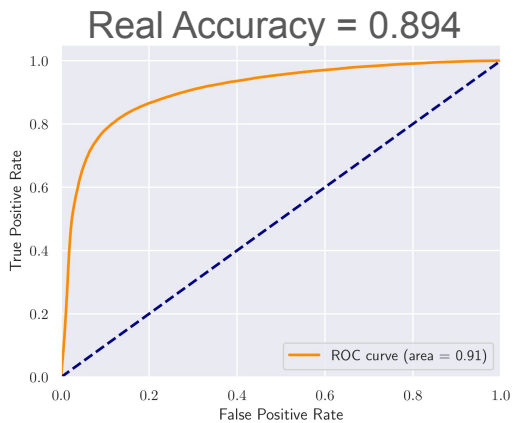
Domain Adversarial Neural Networks on noisy data

We train an DANN: **t = 3, λ = 50**

1. Architecture: Main task 6x10x10x10x5x1, Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.



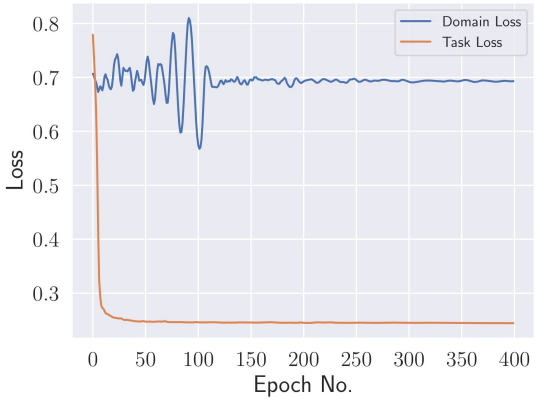


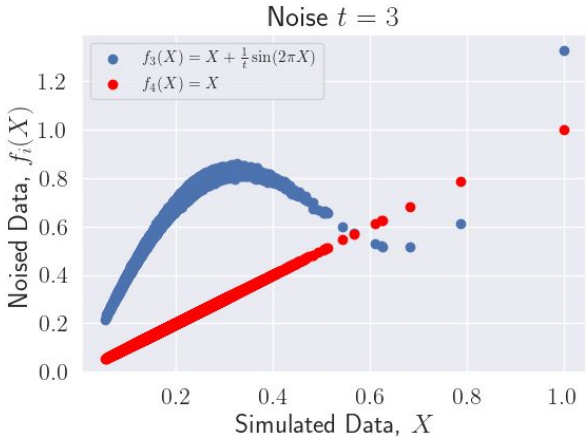


Domain Adversarial Neural Networks on noisy data

We train an DANN: **t = 3, λ = 50**

1. Architecture: Main task 6x10x10x10x5x1, Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.



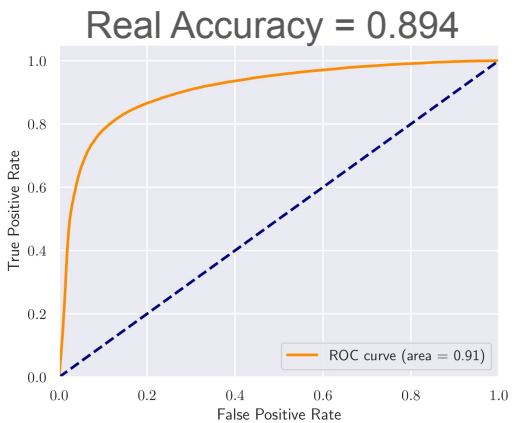


MLP:

Real Accuracy = 0.693

DANN:

Real Accuracy = 0.894



Exploring DANNs further - Multiple Transformations

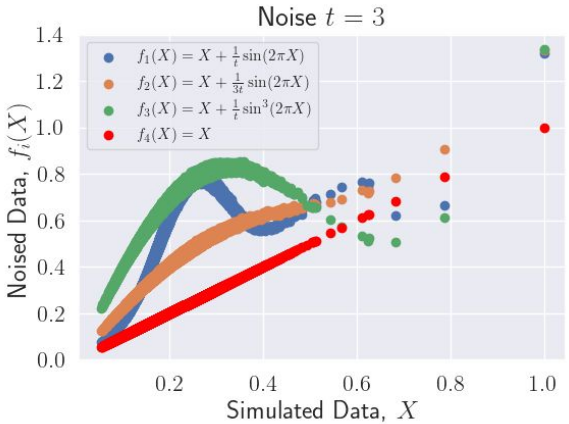
We train an DANN: **$t = 3, \lambda = 150$**

1. Architecture: Main task 6x10x10x10x5x1,
   Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

Exploring DANNs further - Multiple Transformations

We train an DANN: **t = 3, λ = 150**

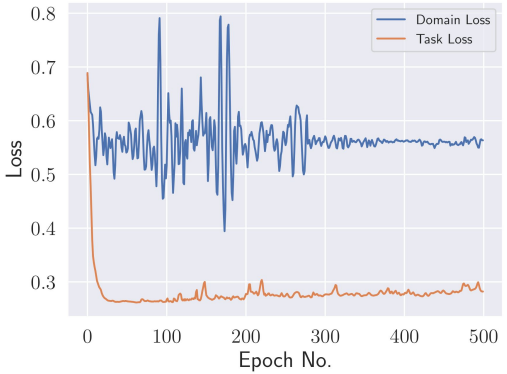
1. Architecture: Main task 6x10x10x10x5x1,
   Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

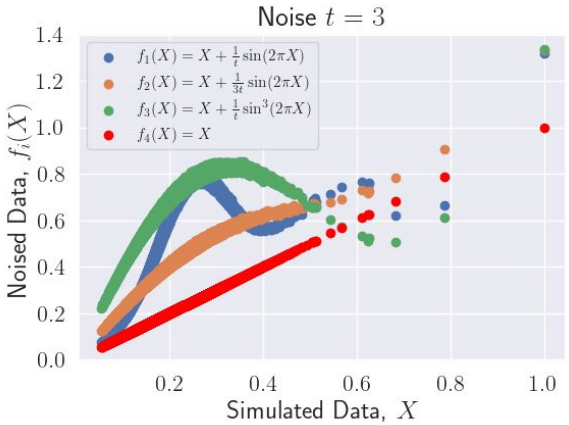


Exploring DANNs further - Multiple Transformations

We train an DANN: **t = 3, λ = 150**

1. Architecture: Main task 6x10x10x10x5x1, Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.

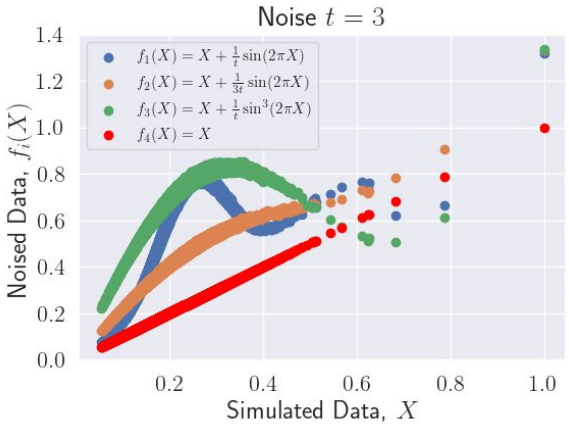


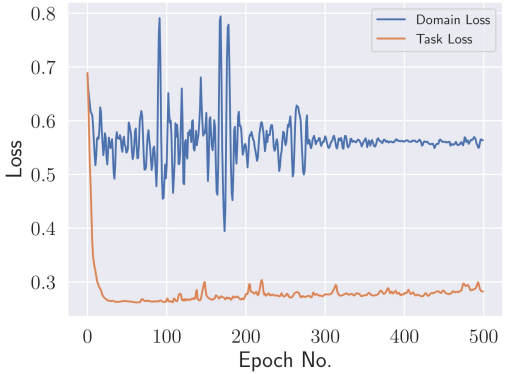


Exploring DANNs further - Multiple Transformations

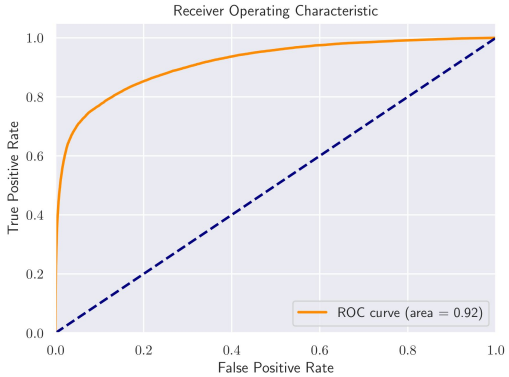
We train an DANN: **t = 3, λ = 150**

1. Architecture: Main task 6x10x10x10x5x1, Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.





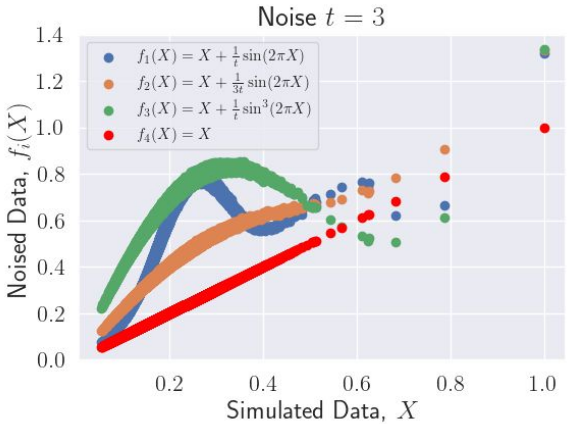
Real Accuracy = 0.899



Exploring DANNs further - Multiple Transformations

We train an DANN: **t = 3, λ = 150**

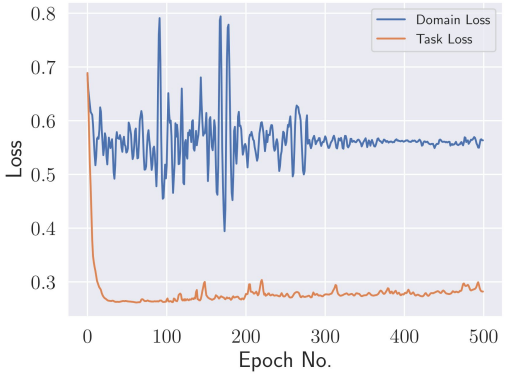
1. Architecture: Main task 6x10x10x10x5x1, Domain Discriminator 10x20x10x1.
2. LR: 0.001 - with scheduler.
3. Validation loss early-stopper.



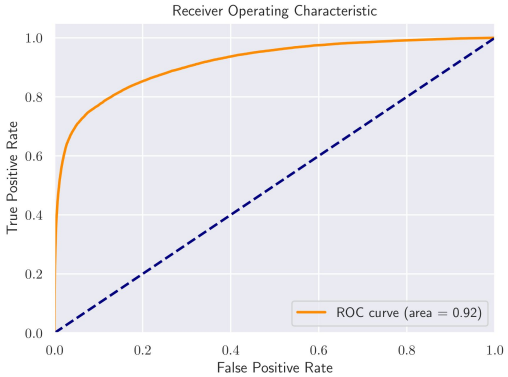
MLP:
Real Accuracy = 0.693

DANN:
Real Accuracy = 0.894

Hybrid-DANN:
Real Accuracy = 0.899

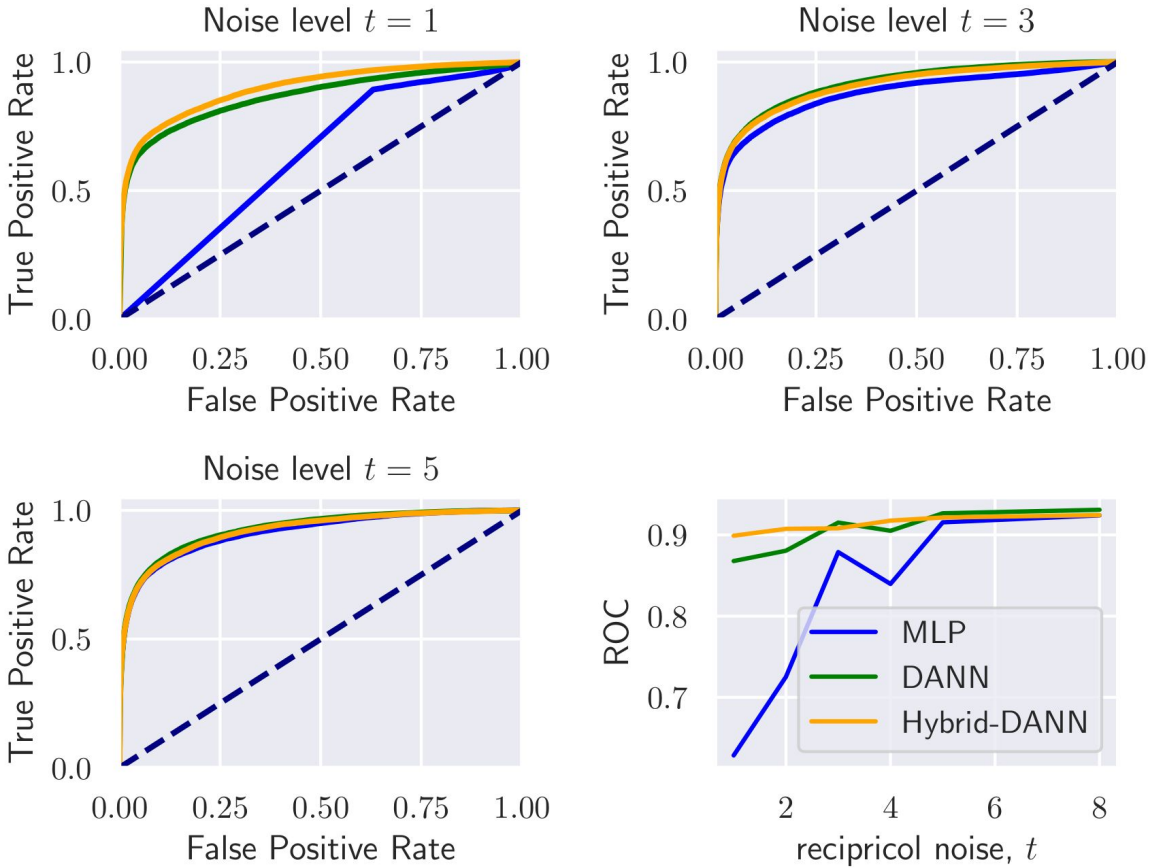


Real Accuracy = 0.899



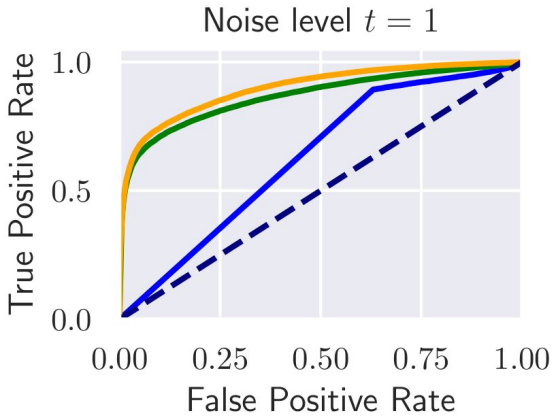
Main Results

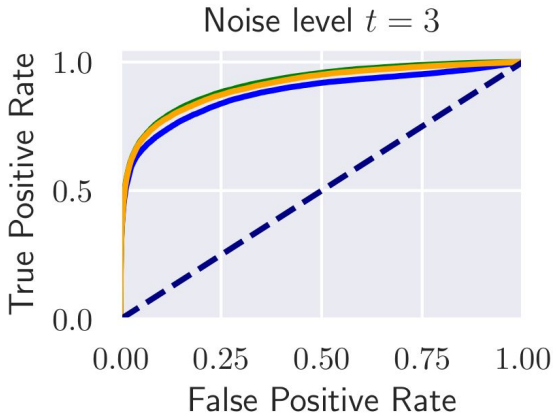
- DANNs provide a methodology to generalize simulated data to real data.

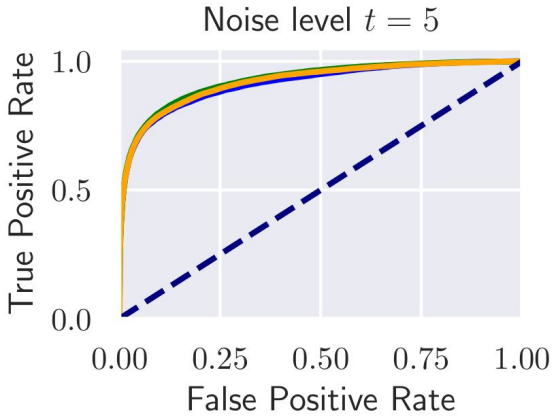


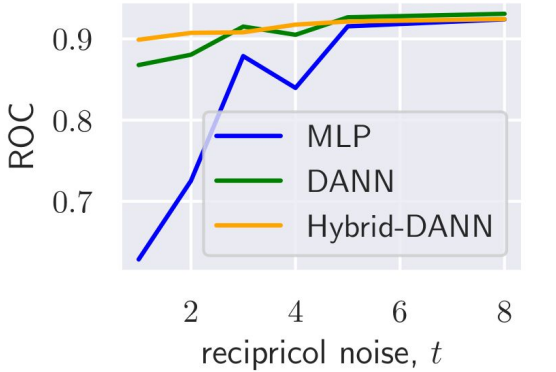
Main Results

- DANNs provide a methodology to generalize simulated data to real data.
- Relative increase in performance increases as quality of simulation decreases.



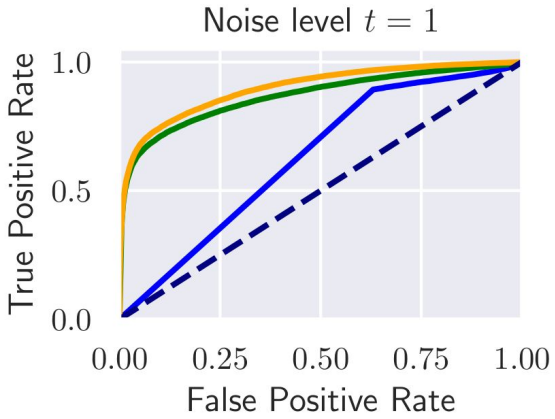


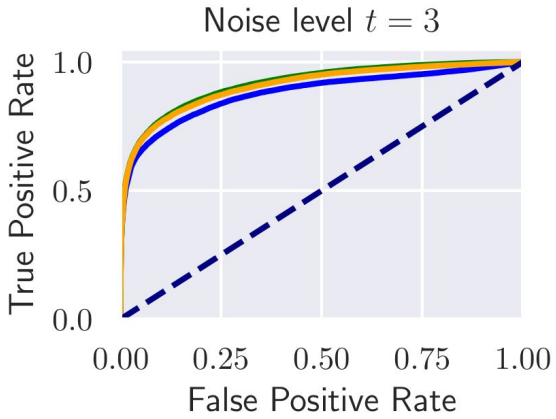


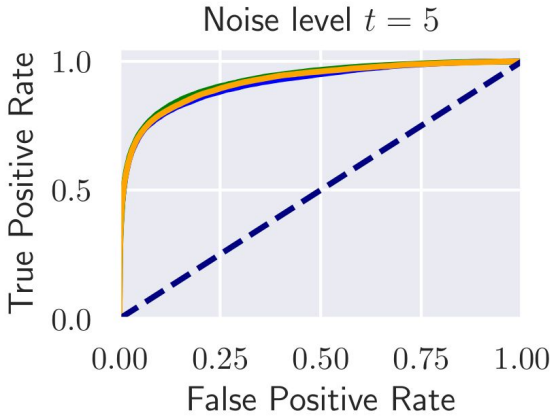


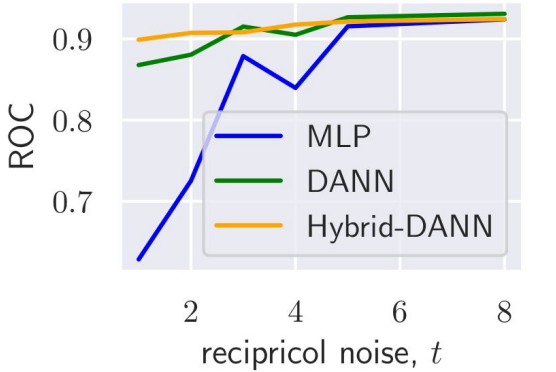
Main Results

- DANNs provide a methodology to generalize simulated data to real data.
- Relative increase in performance increases as quality of simulation decreases.
- Multiple noise profiles can be generalized via a Hybrid-DANN approach, taking advantage of pairwise learning.



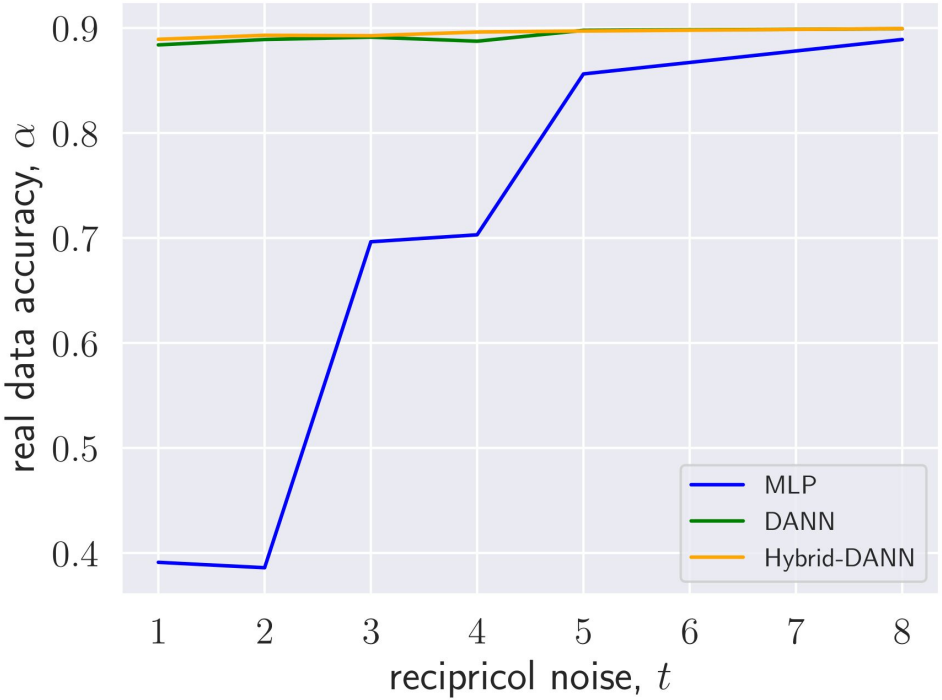






Main Results

- DANNs provide a methodology to generalize simulated data to real data.
- Relative increase in performance increases as quality of simulation decreases.
- Multiple noise profiles can be generalized via a Hybrid-DANN approach, taking advantage of pairwise learning.

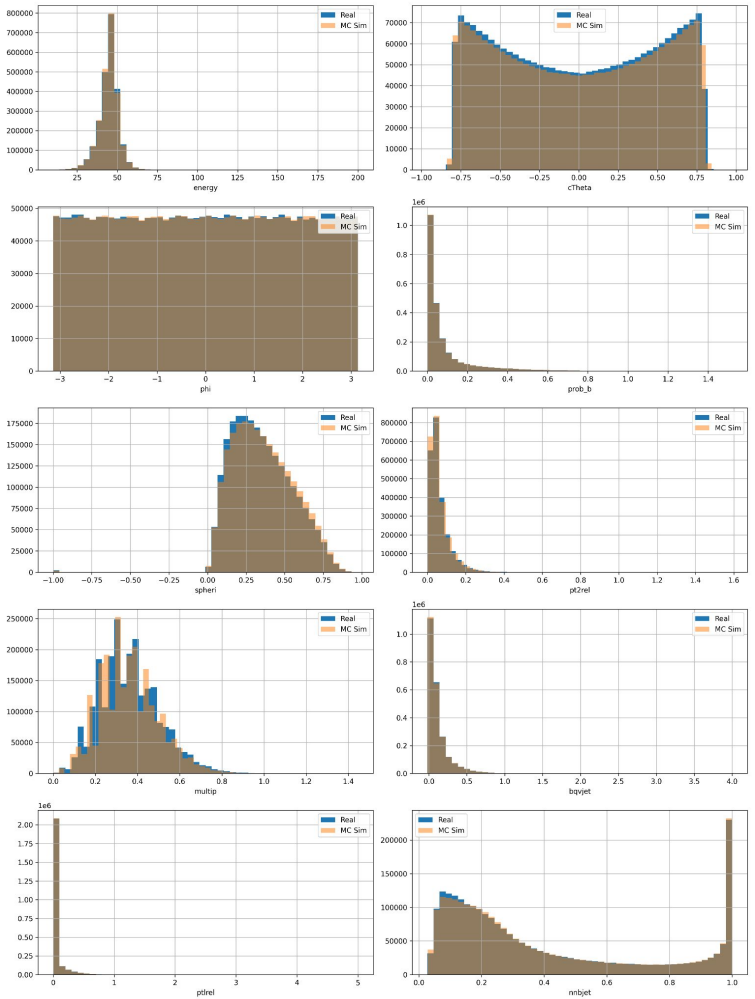


Main Results

We provided theoretical results with theoretical domain shifts.

- The target Aleph Bjet dataset had no domain shift.
- Indistinguishable from MC data.

Consequently, DANNs and Hybrid-DANNs are ineffective, as the fundamental adversarial cannot discriminate.



Thank you for your time (:

Appendix:

Ganin, Yaroslav, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2015. “Domain-Adversarial Training of Neural Networks.” http://arxiv.org/abs/1505.07818

GitHub:

https://github.com/mengsig/APML_DANN

Real Data - Aleph data results

- Unfortunately, due to the lack of a domain shift between the Monte Carlo and real data, DANNs were not applicable. In summary we found for t = 3:

ML-Architecture	Naive MLP	DANN	Hybrid-DANN
Real Data Accuracy	0.853	0.732	0.841

- Showcasing that the Domain Adversarial Neural Networks could not discriminate the real and simulated data. Moreover, the Hybrid-DANN only does better than the DANN due to the pairwise learning that exists in this architecture.
- That however, does not mean that DANNs are not applicable to real data! It just means that they are applicable when there is a real domain shift in the data!

Different DANN backpropagation

- In our initial implementation (see Github **_novel_learning.py files), we treated the entire model as a single entity, and did backpropagation in the following manner.
  - Forward pass of simulated data with full gradient checkpointing.
  - Forward pass of real data without gradient checkpointing in the Label Classifier.
  - Gradient reversal layer between Feature Extractor and Domain Classifier.
  - Full back-propagation based on the sum off

    total_loss = label_loss + lambda * domain_loss

  - This resulted in similar results to the real DANN, but for different reasons, as the Domain Classifier had substantially higher accuracy. Occasionally, this model outperformed the normal DANN.

Consistency metric

- Since we worked with MC data, we had labels for finding accuracy.

Consistency metric

- Since we worked with MC data, we had labels for finding accuracy.
- Real data have no labels, so we need some other way to measure performance.

Consistency metric

- Since we worked with MC data, we had labels for finding accuracy.
- Real data have no labels, so we need some other way to measure performance.
- We use a metric called **consistency:**

$$\text{Consistency} = 1 - \frac{1}{N}\sum_{i=1}^{N}\left|\hat{y}_i - \frac{1}{k}\sum_{j\in k\text{-NN}(x_i)}\hat{y}_j\right|$$

Consistency metric

- Since we worked with MC data, we had labels for finding accuracy.
- Real data have no labels, so we need some other way to measure performance.
- We use a metric called **consistency:**

$$\text{Consistency} = 1 - \frac{1}{N}\sum_{i=1}^{N}\left|\hat{y}_i - \frac{1}{k}\sum_{j \in k\text{-NN}(x_i)}\hat{y}_j\right|$$

Prediction of i-th observation

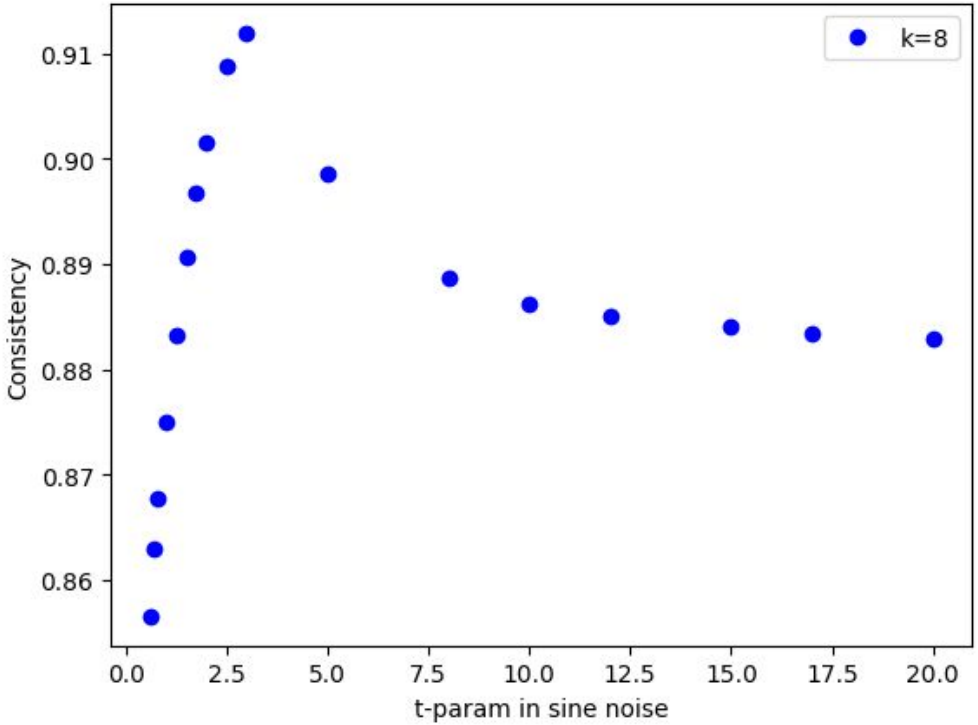
Consistency metric

- Since we worked with MC data, we had labels for finding accuracy.
- Real data have no labels, so we need some other way to measure performance.
- We use a metric called **consistency:**

$$\text{Consistency} = 1 - \frac{1}{N}\sum_{i=1}^{N}\left|\hat{y}_i - \frac{1}{k}\sum_{j \in k\text{-NN}(x_i)}\hat{y}_j\right|$$

Predictions of its
k-nearest neighbors

Consistency metric - sinusoidal transformation



Consistency metric - Gaussian transformation