Disguise Adversarial Networks for Imbalanced Ad Conversion Datasets

Derek Zhao, James F. Xue, Chaitanya Dara, Sai Pavan Kumar Unnam, Daniel Gomez

1 Abstract

A common issue in using machine learning to predict ad conversions in click-through rate datasets is the imbalanced classification problem: we wish to label a data point as belonging to either a negative or positive class, but the amount of samples from one class significantly outnumbers that of the other. Under such conditions, binary classifiers struggle to train effectively due to the lack of sufficient exposure to minority class samples. This is especially true in click-through rate datasets where the positive class typically comprises between 0.1% to 1% of the training data. An experimental architecture for addressing class imbalance, the Disguise Adversarial Network (DAN), disguises negative samples as positive ones to supplement the minority class. [2] We successfully implemented a DAN and tested it extensively on two different MediaMath-provided datasets. The DAN, while showing some promise on datasets with extreme class imbalance, ultimately does not perform better than simpler solutions such as naïve oversampling.

2 Introduction

MediaMath is a leader in programmatic marketing technology that connects marketers to consumers, typically through using machine learning to predict which ads are most likely to be clicked on by which consumers. By recommending to consumers ads they are more likely to click, MediaMath is able to improve its click-through rate (CTR) as well as its revenue.

A pervasive problem encountered when applying machine learning models to CTR datasets is that of imbalanced class labels. Typically, positive class examples (when a consumer clicks on the displayed ad) comprise only 0.1% to 1% of the dataset. This paucity of positive samples prevents binary classifiers from more effectively distinguishing positive from negative samples and significantly hinders the models' performance.

Many methods have been proposed to address the detrimental effects of class imbalance, of which the most common include:

- **Minority class oversampling**: augmenting the dataset with extra copies of positive class samples
- **Synthetic minority oversampling technique (SMOTE)**: generating synthetic positive class samples from weighted averages of original positive class samples [1]
- **Adaptive Synthetic Sampling Approach (ADASYN)**: a variation on SMOTE that adds Gaussian noise to the synthetic positive sample [3]

Researchers at Samsung Research America [2] recently proposed a novel architecture for handling class imbalance. Drawing inspiration from the surprising effectiveness of Generative Adversarial Networks (GANs), they propose a method for transforming negative samples into positive samples, dubbed the Disguise Adversarial Network (DAN).

3 Disguise Adversarial Networks

Whereas SMOTE seeks to address class imbalance by using true positive samples to generate synthetic positive samples, DANs "disguise" true negative samples as positive samples. Deng et al. argue that SMOTE is severely limited in that it assumes the available true positives sufficiently span the complete space of all possible positive samples [2]. Rather, an upsam-

pling technique that uses the abundance of negative class samples to produce new positive samples offers a much wider variety of minority class examples and helps to train a more robust classifier.

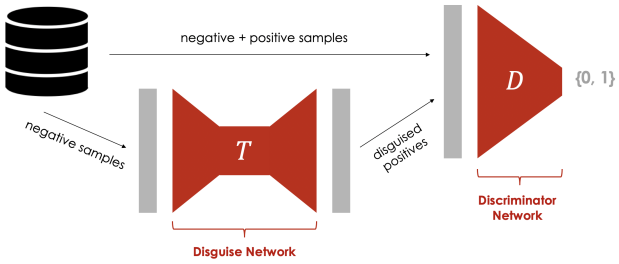

Figure 1: **DAN architecture**

The DAN consists of two densely connected networks (Figure 1). The discriminator network D is the binary classifier tasked with distinguishing between negative and positive examples, and it is on D that we ultimately evaluate the effectiveness of the DAN. The disguise network T attempts to transform negative samples in such a way that D is more likely to classify the transformed samples as positive. Note that without T, the architecture is no different from an ordinary binary classifier.

To encourage T to properly disguise negative samples capable of fooling D into making positive classifications, T is trained through minimizing the following loss function $\mathcal{L}_1$:

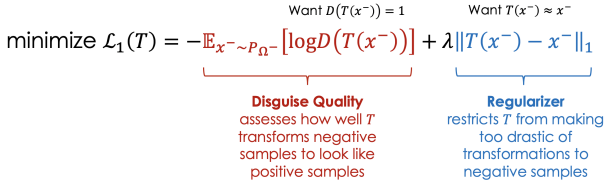

Figure 2: **The disguise loss**

where:

- x^- denotes a negative sample
- $T(x^-)$ denotes a negative sample disguised as a positive sample
- $D(T(x^-)) \in [0, 1]$ denotes the discriminator's confidence that the disguised sample belongs to the positive class

The regularization term serves two interconnected purposes. First, it restricts T from making transformations that are too drastic, the reasoning being that a disguised sample should retain as many of its original features as possible so that the set of synthetic positives are more diverse than the set of original positives. Second, and perhaps more practically, it prevents T from collapsing and outputting the same vector regardless of input. Otherwise, T would simply transform all negative samples into a single disguised sample with the highest probability of being classified as positive.

To motivate the final formulation of the discriminator loss $\mathcal{L}_2$ that we minimize to train the discriminator network, we introduce a seemingly reasonable but more naïve discriminator loss function. We assign all disguised samples a negative label, and simply minimize the binary cross-entropy loss:

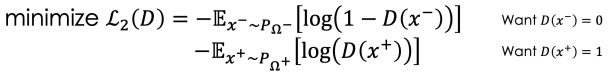

Figure 3: **A naive discriminator loss**

Unfortunately, this naïve discriminator loss poses a major problem: T may sometimes very successfully disguise a negative sample into a convincing positive sample, and encouraging a well-disguised sample to be classified as negative would confuse D and hurt its performance.

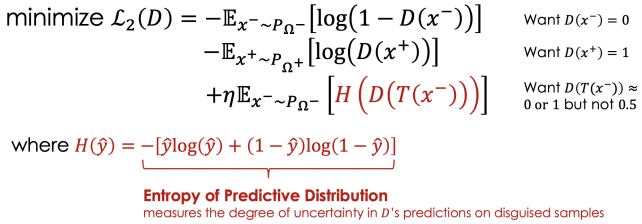

Figure 4: **The discriminator loss**

Instead, the authors suggest only calculating the binary cross-entropy loss on originally labeled samples and leaving the disguised samples unlabeled. The discriminator's predictions on these unlabeled samples are used to calculate an average predictive distribution entropy, whose sum with the average binary cross-entropy forms the discriminator minimization objective [2]. In other words, we choose not to impose a label on the disguised samples because we acknowledge that some disguises will be effective while others will not. All we ask of D is that when it makes a prediction on disguised data, it is confident about its prediction.

	CTR	CPC
disk size (GB)	9.7	1.8
# training samples	25,021,860	3,834,011
# features	76	97
% positive class	0.087%	10.46%

Table 1: **Comparison of CTR and CPC datasets**

4 Datasets

MediaMath has provided us with data from two marketing campaigns, designated the CTR (click-through rate) and CPC (cost-per-click) datasets on which to evaluate DAN performance. For both campaigns, each row consists of a set of features describing the ad, the conditions under which it was displayed, and the user it was shown to. A label accompanies each row, where 1 indicates that the user clicked the ad and 0 otherwise. Despite the names of these datasets, they have very similar feature sets and targets.

Table 1 summarizes the key differences between the CTR and CPC datasets. The CTR campaign is particularly challenging in that the proportion of positive class samples is much smaller than that of the public datasets used to benchmark the DAN. Furthermore, at 9.7GB in uncompressed training set size, we require cloud resources to complete the computationally intensive tasks of processing and modeling the data.

5 Implementation

5.1 Data processing pipeline

An especially difficult aspect of both datasets is the extremely high cardinality of many of their categorical features, some of which are reported in Table 2. To address these issues, two data processing pipelines were developed to serve MediaMath data to our DAN implementation (discussed in later subsections), with each pipeline providing a different representation of the data as well as a different means of supplying the data. We refer to the pipelines as the 1) embedding pipeline and 2) one-hot pipeline.

5.1.1 Embedding pipeline

One method of dealing with high-cardinality categorical features is to ask the DAN, in addition to performing positive class augmentation, to also learn a

	CTR	CPC
site_id	87,271	49,475
category_id	2,559	2,265
hashed_app_id	1	3,885
device_model	1,222	915
browser_version	985	652
deal_id	801	268
dma_id	212	211

Table 2: **Cardinality of features in CTR and CPC datasets**

low-dimensional dense representation of each categorical value, not unlike in Word2Vec or GloVe. To facilitate such learning, features to be represented as an embedding are typically index-encoded, with each unique value mapping to a unique integer that functions as the lookup index of an embedding matrix. Thus, the transformed data is compact compared to an alternative such as one-hot encoding, and a fully in-memory transformation process is feasible.

Broadly speaking, the embedding pipeline loads an entire raw dataset from disk into RAM, performs a set of transformations using properties learned from the training set, and writes the processed dataset back to disk. When training the DAN, the entire processed training and validation sets are loaded back into memory for the model to fit on in batches.

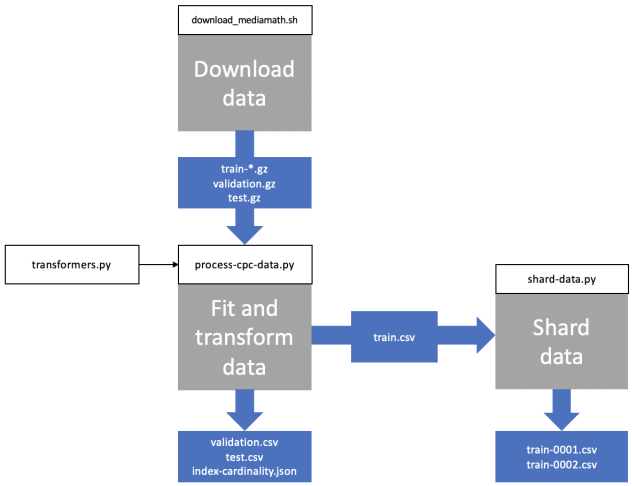

Figure 5: **An embedding-based pipeline**

As shown in Figure 5, raw data is downloaded as a series of GZIP files. The **fit and transform** stage of the pipeline decompresses the data and fits a set of custom scikit-learn transformers on the training

data and applies them to all data, resulting in the following:

- A set of user-specified features (such as **column_weights**) are dropped.
- All continuous features are imputed with 0.
- All continuous features are standardized.
- All categorical features are imputed with -1.
- All categorical features with cardinalities above 20 are index-encoded.
- All categorical features with cardinalities below or equal to 20 are onehot-encoded.

The fit and transform stage outputs a single CSV file per dataset split (train, validation, and test) as well as a JSON mapping the column index of each categorical feature to its cardinality (**index-cardinality.json**). The latter is crucial for transforming categorical features into their embedded representations and described more fully in the next subsection. The training data is further sharded to facilitate downstream experiments in which we wish to alter its class distribution.

5.1.2 One-hot pipeline

The one-hot pipeline was developed to address many of the limitations inherent to the embedding pipeline's fully in-memory transformation process. While non-sparse one-hot transformations typically require prohibitive amounts of memory and disk space, this pipeline achieves scalability through two means:

1. Serving data in batches to the DAN for lower memory consumption
2. Processing data only when served for lower disk usage

The first mechanism is achieved through Tensorflow's CsvDataset API while the second through Tensorflow's FeatureColumns API. Both mechanisms are encapsulated within a class called **DataGenerator** for ease of use when interfacing with the DAN.

As shown in Figure 6, the pipeline begins with downloading the raw datasets as GZIP files. In the **fit and shard** stage, JSON metadata is created for each feature that provides instructions for how to transform the data when served by the data generator. In particular, **numerical-stats.json** contains the mean and standard deviation of each continuous feature so that they may be standardized in batches later, and **categorical-vocab.json** contains the vocabulary list of each categorical feature so that they may be one-hot encoded in batches later.

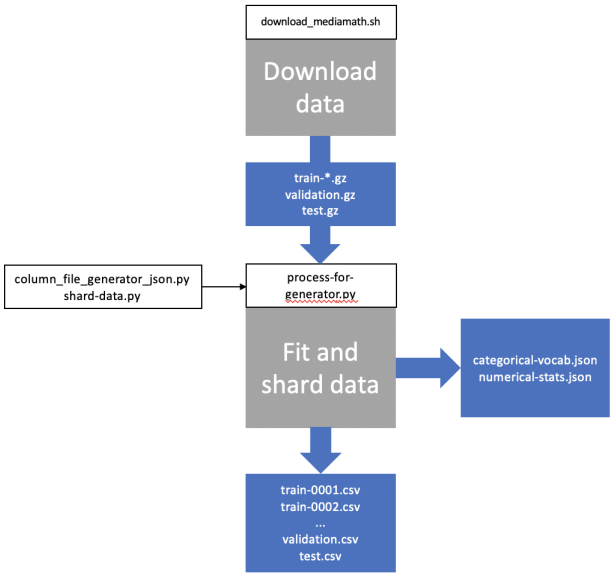

Figure 6: **A one-hot-based pipeline**

An important implementation detail is that because the training data was originally separated into CSVs containing exclusively negative class data and exclusively positive class data, the training data must be sharded into much smaller CSVs to facilitate random shuffling of negative and positive class data within the data generator.

5.2 Model implementation

We implemented our DAN model using Tensorflow's graph-based API and focused on an object-oriented architecture. By implementing the DAN as a self-contained class, we abstracted away the unintuitive notions of Tensorflow sessions and metagraphs, facilitating ease of use for downstream experimentation and tuning. Figure 7 illustrates how the core components of the DAN library interact.

The disguise network and discriminator network classes both inherit from a common network base class, and a disguise network object and discriminator network object are passed as arguments to the constructor for the DAN. Such a setup allows the user to freely customize the disguise and discriminator networks independently of each other and the overall DAN framework, as shown in Figure 8.

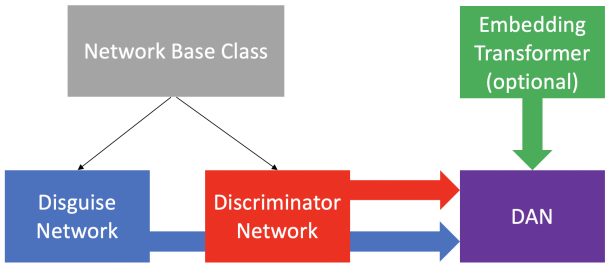

Figure 7: **An object-oriented implementation of the DAN**

```
disguise = Disguise(NUM_FEATURES, [4, 6, 4])
discriminator = Discriminator([4, 4])
dan = DAN(disguise, discriminator, CHECKPOINT_DIR, LOG_DIR)
```

Figure 8: **Defining a DAN**

Both the disguise and discriminator networks support any dense network topology as well as user-specified choices for activation function, batch normalization, and dropout regularization.

When the DAN is initialized without a disguise network, it behaves identically to a regular densely-connected classifier. It is through this mechanism that we measure baseline performance.

5.2.1 Embeddings for high cardinality features

The **EmbeddingTransformer** class facilitates model ingestion of index-encoded data produced by the embedding pipeline. This is a modification to the DAN framework that Deng et al. did not need to explore due to the much lower cardinality of their datasets. While embedding transformations are a common technique for neural classification tasks, in the context of a DAN, we must be careful about when the transformation is applied.

Embedding samples of data after they have been disguised (Figure 9) is not fruitful because this would require the disguise network to attempt to transform index-encoded categorical data. Such an encoding is devoid of any ordinal meaning and thus would be problematic for the disguise network to transform. Instead, we embed all data before they flow into the disguise or discriminator network (Figure 10), thus requiring the disguise network to create transformed samples in embedded space, and plan the architectures of both networks according to the data's dimensionality in embedded space.

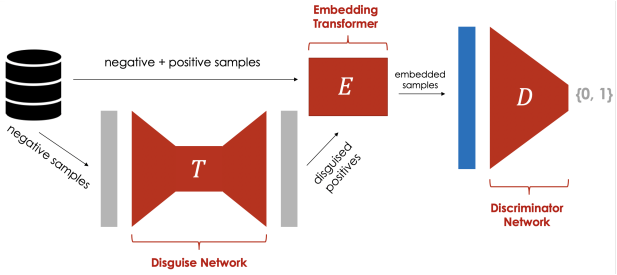

Figure 9: **Incorrect usage of the embedding transformer**

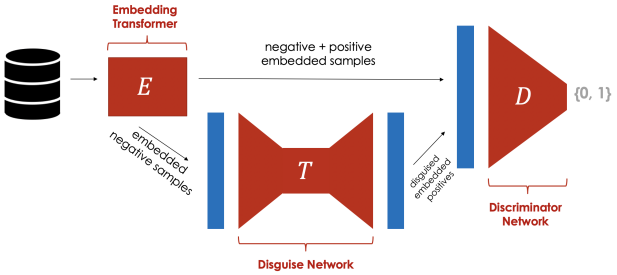

Figure 10: **Correct usage of the embedding transformer**

One final implementation detail is that we only allow gradients from the discriminator network to affect how the embeddings are updated during training, as the disguise network should not have any bearing on how the embeddings are learned.

```
embedding_transformer = EmbeddingTransformer(index_map=index_map)

disguise = Disguise(
  num_inputs=embedding_transformer.calc_num_outputs(num_inputs),
  hidden_nodes=[16, 8, 8, 16])

discriminator = Discriminator(
  hidden_nodes=[16, 8])

dan = DAN(
  disguise, discriminator, CHECKPOINT_DIR, LOG_DIR,
  embedding_transformer=embedding_transformer,
  num_inputs=num_inputs)
```

Figure 11: **DAN usage with embedding layers**

Because not all features are categorical or index-encoded, the embedding transformer requires instructions (provided in the form of a map) as to which columns of input data are index-encoded and the cardinality of said index-encoded features. From the cardinality, the embedding transformer calculates the dimensionality of the embedding to be applied. In general, we set the embedded dimensionality of a feature

to be the ceiling of the cube root of the feature's cardinality.

5.2.2 Usage

The DAN class exposes only three in-memory methods to the user, where X and y are NumPy arrays fully loaded into memory:

1. **fit(X, y)**: calls internal partial fit methods to iteratively train the DAN
2. **predict(X)**: outputs a predictive distribution for each sample
3. **transform(X)**: outputs a disguised version of each sample

To allow the model to train and predict on datasets of arbitrarily large sizes, the following functions accept instances of the **DataGenerator** class:

- **fit_generator(train_datagen, val_datagen)**: calls internal partial fit methods to iteratively train the DAN and evaluate the model's performance on validation data.
- **predict_generator(test_datagen)**: calls internal partial predict methods to infer predictions.
- **predict_proba_generator(test_datagen)**: calls internal partial predict_proba methods to infer predictive distributions.
- **evaluate_generator(test_datagen)**: returns the model's cross entropy loss, accuracy, AUROC.

5.2.3 Tensorboard and metric reporting

To aid in model debugging and monitoring during training runs, we modified the DAN to report metrics to Tensorboard. Table 3 provides details: most metrics are calculated each training step from the training set, but a handful are calculated every epoch, and two are evaluations on the validation set.

- **cross entropy**: Measures the discriminator's performance on labeled training data. Lower is better.
- **predictive entropy**: Measures the discriminator's confidence when classifying unlabeled data. Lower is generally better, but too low a value also suggests the disguise network has collapsed and is not performing useful data augmentation.

metric	evaluation	frequency
cross entropy	train	step
predictive entropy	train	step
discriminator loss	train	step
disguise quality	train	step
regularizer	train	step
disguise success rate	train	step
disguise loss	train	step
train accuracy	train	epoch
train AUROC	train	epoch
validation accuracy	validation	epoch
validation AUROC	validation	epoch

Table 3: **Tensorboard metrics**

- **discriminator loss**: Weighted sum of cross entropy and predictive entropy. Lower means the discriminator network is learning.
- **disguise quality**: Measures the effectiveness of the disguise network at transforming negative samples into positive-seeming samples (from the perspective of the discriminator network). Higher is better, but too high a value could indicate disguise network collapse.
- **regularizer**: Measures how much freedom the disguise network is exercising when transforming negative samples into disguised samples. Too low a value suggests the disguise network is behaving more like an autoencoder, yet too high a value suggests the disguise network is more likely to produce less diverse disguises.
- **disguise success rate**: A more interpretable proxy for **disguise quality**, the proportion of negative samples in a batch that, after being disguised, are classified as positive by the discriminator network.
- **disguise loss**: Weighted sum of negative disguise quality and the regularization term. Lower means the disguise network is learning.
- **train accuracy**: Accuracy of the entire training set.
- **train AUROC**: Area under ROC curve for the entire training set.
- **validation accuracy**: Accuracy of the entire validation set.
- **validation AUROC**: Area under ROC curve for the entire validation set.

With the availability of validation metrics for Tensor-

board logging, the model only checkpoints when its validation AUROC improves over its previous optimal validation AUROC and ceases training if no improvement in optimal validation AUROC is detected after four epochs of training.

5.3 Challenges

A significant setback we encountered in adapting the DAN to work with custom data generators is that while a saved and loaded model evaluates correctly when using the in-memory functions for the embedding pipeline, it does not do so when using the generator-based functions for the one-hot pipeline. The processed data output by a data generator has a slightly different column ordering depending on whether a new DAN is instantiated (as when training) or a saved DAN is loaded from its checkpoint (as when evaluating). Attempts to pinpoint the source of this error were unsuccessful, so as a workaround, we immediately evaluate a model after early stopping concludes its training run, using the model's most recent state rather than that checkpointed with the highest validation AUROC.

An additional issue that limited the success of the one-hot pipeline was that our DAN implementation does not use sparse matrices, save for in the embedding transformer. Thus, the DAN incurs significant performance penalties when ingesting high-dimensionality data, as with MediaMath's datasets which explode to well over 100000 dimensions when one-hot encoded. These penalties are threefold given that they occur in three areas of the DAN architecture:

- The weights connecting the input layer of the disguise network to its first hidden layer.
- The weights connecting the final hidden layer of the disguise network to its output layer.
- The weights connecting the input layer of the discriminator network to its first hidden or output layer.

Though we were able to evaluate the DAN using the one-hot pipeline, for the above reasons, the model took prohibitively long to train, and the hyperparameter search was not as extensive as that for the embedding pipeline.

6 Results

We trained and evaluated the DAN on synthetic data as well as MediaMath's CPC and CTR datasets. In the following subsections, we describe our experimental setup and analyze the performance of the DAN across a range of conditions.

6.1 Synthetic data

As an initial proof of concept, we trained a small DAN on two-dimensional synthetic data where samples for positive and negative classes were drawn from different Gaussian distributions. Consisting of 90000 negative class samples and 10000 positive class samples, the synthetic data was useful for unit testing and visualizing the behavior of the DAN. Figure 12 illustrates the distribution of data for the two classes.

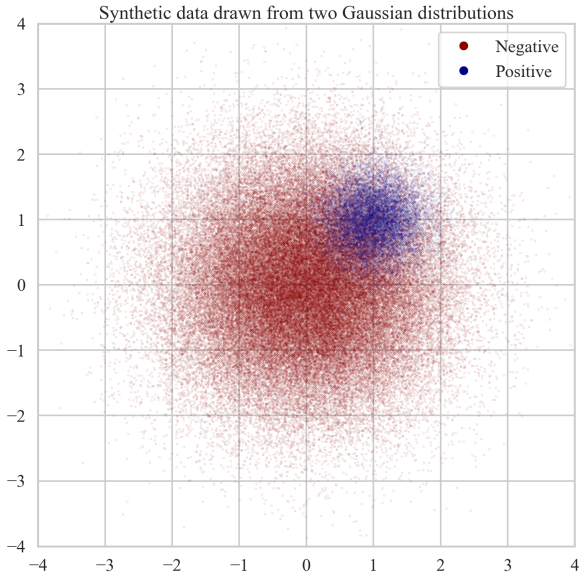

Figure 12: **Synthetic data**

Because the data is drawn from Gaussian distributions, we first trained and tested a Gaussian Naive Bayes classifier to approximate Bayes' optimal error, providing an upper bound on how much a DAN could improve on baseline neural network models. Figure 13 shows the distribution of 20 validation accuracies for each model. The **baseline** and **discriminator** models are conceptually the same, a densely-connected binary classifier, but with the former implemented in Keras and the latter implemented in Tensorflow as part of the DAN library. Therefore, the differences in their accuracy distributions is attributable to noise and unfortunately suggests that

the DAN's seemingly better median accuracy could be statistically insignificant.

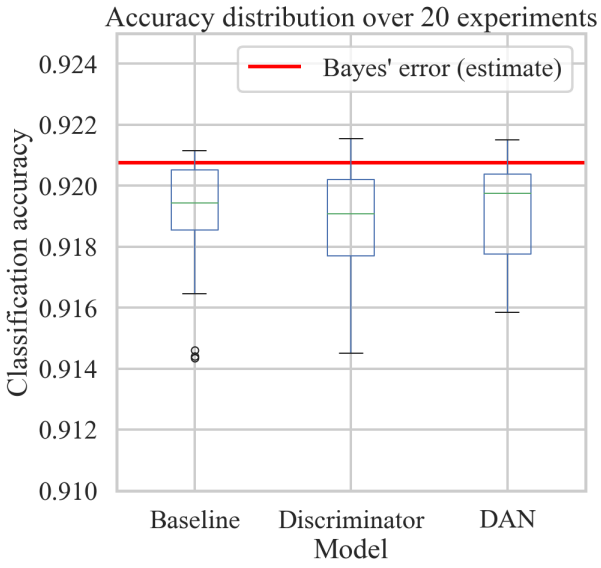

Figure 13: **Validation accuracy on synthetic data**

While the two-Gaussian classification problem might be too simple to yield conclusive results about the DAN's effectiveness, it does lend itself to understanding the disguise network's behavior. As previously discussed, the disguise loss contains a regularization term that penalizes the disguise network for making too drastic of transformations on the incoming negative samples. The strength of this regularization is controlled through hyperparmeter λ. We trained the DAN for different settings of λ as well as different choices of activation functions, disguised every negative sample in the synthetic dataset, and visualized the disguised data (Figure 14).

As expected, a high value for λ prevents the disguise network from applying any meaningful transformation to the negative data, while a low value such as $\lambda = 0$ causes the transformation to collapse towards a tiny region. Rather surprising, however, is the disguise network's behavior for moderate values of λ. Deng et al. claim that the purpose of the regularization term was to prevent drastic transformations to the data, but in the two-dimensional case, it appears that λ not only controls the extent of the transformation but also the proportion of samples that are altered.

In our experiments, we observed that settings for activation function and λ that transform samples into both negative and positive regions performed better than those that transformed samples into exclusively positive regions. A potential explanation may be that transforming negative samples onto and near the decision boundary between negative and positive classes forces the model to clarify its decision boundary.

6.2 CPC data

Given the manageable size of the CPC dataset, we evaluated the DAN using both the embedding and one-hot pipeline and, for each pipeline, compared its performance against some subset of the following baselines:

- **Logistic regression**: No disguise network is used for positive class augmentation.
- **Neural network**: No disguise network. The discriminator network is a 2-layer DNN.
- **Logistic regression with undersampling**: No disguise network. In the training split, the negative class data is undersampled so that the class distribution is 50% negative class and 50% positive class. This is the current MediaMath baseline.
- **Neural network with undersampling**: No disguise network. The discriminator network is a 2-layer DNN. In the training split, the negative class data is undersampled so that the class distribution is 50% negative class and 50% positive class.

Table 4 summarizes the one-hot pipeline test performance of each baseline along with the best-performing DAN model, and Table 5 does the same for the embedding pipeline. For the one-hot pipeline, the two types of discriminator architectures used were either a logistic regression or a DNN with a hidden layer architecture of **128, 32**. For the embedding pipeline, the two types of discriminators used were either a logistic regression or a DNN with a hidden layer architecture of **64, 32**. For both pipelines, when we used a disguise network, it was with **4 hidden layers of 256 neurons each**.

While using the original training data offers significant performance improvements over downsampling the negative data to balance the class distribution, we were disappointed to find that the inclusion of a DAN for positive class augmentation did not help for either the embedding or one-hot data representation. The DAN does not appear to offer benefits in tasks where the data is linearly separable or where base models already predict with high recall.

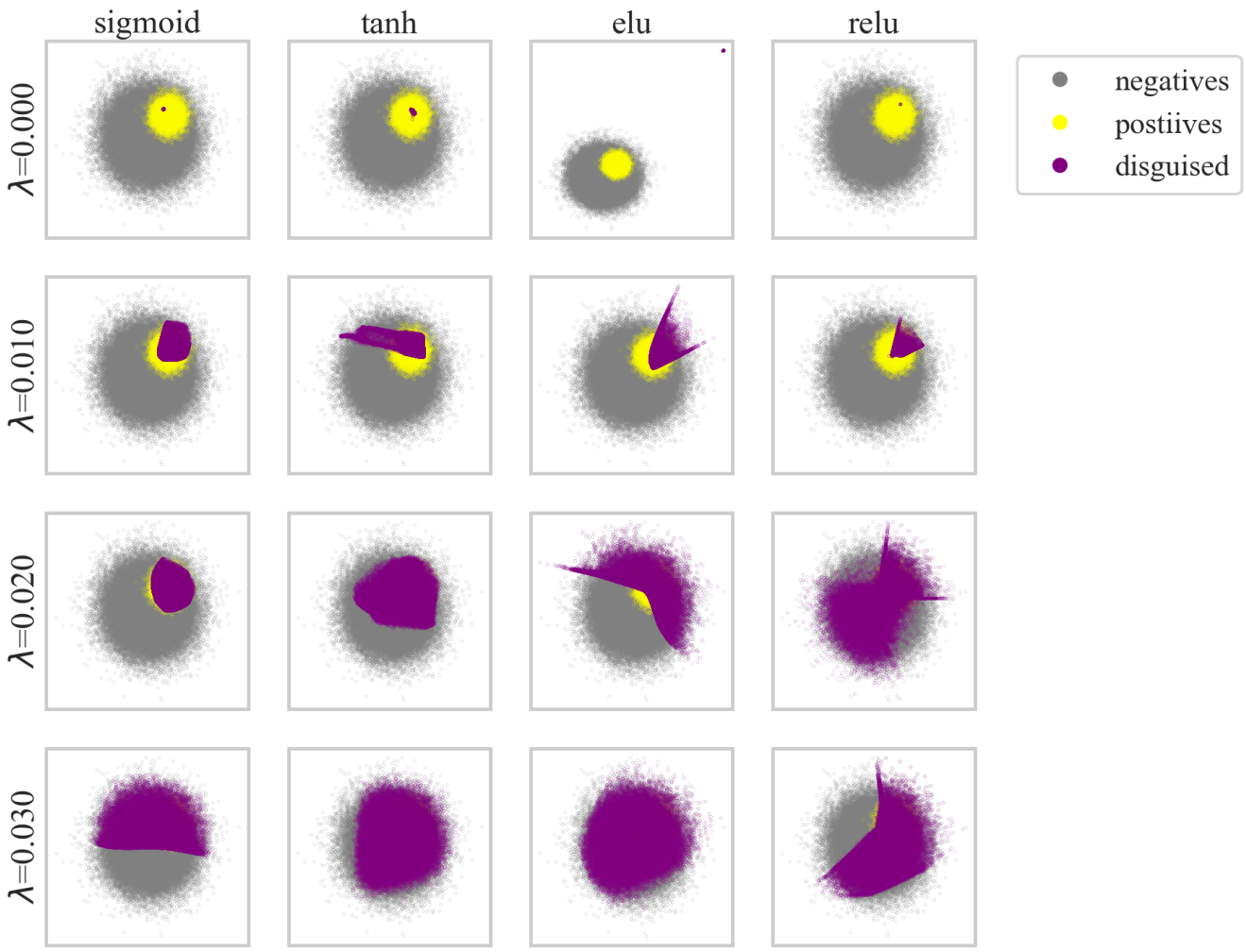

Figure 14: **Disguise transformations by activation and λ**

To account for this, we further skew the CPC training data from 11% positive to 1.1% positive while leaving the validation and test sets unchanged. We ran a new battery of experiments with this highly skewed training data and added a new baseline where we oversample the positive data (through duplication) to bring the training data positive class proportion from 1.1% up to 11%. For these experiments, multiple trials were conducted to account for the random initialization of neural networks and the average test metric is reported.

From Table 6, we see that with artificially imbalanced data, the DAN offers slight improvements in accuracy, AUROC, and recall over baseline models. However, even just modest oversampling from the reduced pool of positive data to recreate the original class distribution far outperforms the DAN.

6.3 CTR Data

The DAN does not fare much better on the CTR dataset (Table 7), where its severe class imbalance should have played to the DAN's strengths. While training on the full dataset produces a better AUROC than doing so with a balanced, undersampled version, using a DAN on the full dataset provides no further improvement.

That the AUROC can be so high despite a precision and recall of 0 is unintuitive but possible; the predicted probabilities show meaningful separation between negative and positive class samples, just not with a classification threshold of 0.5. However, these seemingly inflated AUROC scores bring to light a different issue: like accuracy, AUROC is a fundamentally unsuitable metric for imbalanced classification

positive class proportion	disguise	discriminator	accuracy	AUROC	precision	recall
0.11	no	logistic	0.9163	0.9843	0.5760	0.9852
0.11	no	128 32	**0.9176**	**0.9845**	**0.5806**	0.9800
0.11	256×4	128 32	0.9166	**0.9845**	0.5775	0.9789
0.5	no	logistic	0.9155	0.9437	0.5729	**0.9939**

Table 4: **CPC one-hot pipeline test set performance**

positive class proportion	disguise	discriminator	accuracy	AUROC	precision	recall
0.11	no	logistic	**0.9173**	**0.9844**	0.5795	0.9789
0.11	no	64 32	**0.9173**	0.9843	**0.5800**	0.9754
0.11	256×4	64 32	0.9168	**0.9844**	0.5794	0.9653
0.5	no	logistic	0.9155	0.9650	0.5732	0.9901
0.5	no	64 32	0.9155	0.9668	0.5728	**0.9933**

Table 5: **CPC embedding pipeline test set performance**

positive class proportion	disguise	discriminator	accuracy	AUROC	precision	recall	trials
0.011	no	logistic	0.9002	0.9796	0.6206	0.3054	4
0.011	no	64 32	0.9024	0.9800	**0.6363**	0.3188	5
0.011	256×4	64 32	0.9073	0.9813	0.6319	0.4640	5
0.11 (oversample positives)	no	64 32	**0.9175**	**0.9840**	0.5810	**0.9706**	5

Table 6: **CPC embedding pipeline with artificial imbalance, test set performance**

positive class proportion	disguise	discriminator	accuracy	AUROC	precision	recall
0.00087	no	logistic	0.9988	0.9988	0.0	0.0
0.00087	no	64 32	0.9988	0.9988	0.0	0.0
0.00087	128×4	64 32	0.9988	0.9989	0.0	0.0
0.5 (undersample negatives)	no	logistic	0.6556	0.7334	0.0011	0.3123
0.5 (undersample negatives)	no	64 32	0.6752	0.7323	0.0012	0.3311

Table 7: **CTR embedding pipeline test set performance**

problems because it too is affected by class imbalance.

Consider the ROC curve: its axes are the true positive rate and false positive rate. In a dataset with high class imbalance, overly aggressive models are rewarded because they can improve the true positive rate with minimal increase in the false positive rate due to the extremely high number of negative samples. We chose to use the AUROC since this is the metric Deng et al. optimized for, though in hindsight, the average precision would have been a more meaningful metric.

7 Reflections

The results reported were neither expected nor hoped for, and as such, we cannot conclude that the DAN is an effective solution for MediaMath's datasets. There exist four possible explanations for why our DAN benchmarks underperformed expectations:

1. The DAN was not tuned correctly.
2. Our DAN implementation is inconsistent with that of the original authors.
3. The MediaMath datasets are too different from the original benchmark datasets.
4. The DAN does not work.

The discussions in the following subsections expand on these points.

7.1 Hyperparameter tuning

Like GANs, the DANs were notoriously difficult to tune. A frequently encountered problem with GANs is that, without an optimal hyperparameter setting, the generator network is prone to collapse; it outputs only one sample regardless of the noise vector with which it is seeded. We faced an analogous obstacle when tuning the DAN; the disguise network is prone to both "negative collapse" and "positive collapse."

Negative collapse tends to occur when λ is set too high, thus restricting the freedom the disguise network has to transform incoming negative data when attempting to disguise them. Figure 15 shows how certain metrics and loss function components unfold over a training run on embedded CPC data where $\lambda = 5$. Because of the restrictions a high value for λ places on the disguise network, the disguise quality is pushed ever lower and the disguise success rate

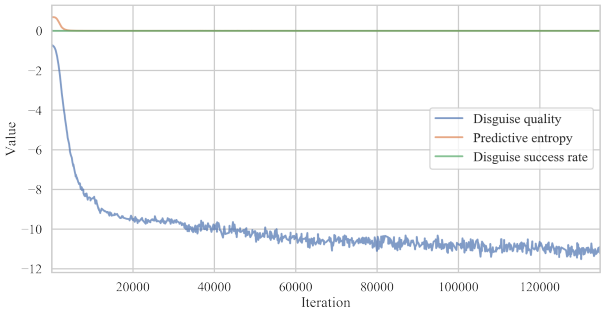

Figure 15: **Disguise network metrics at $\lambda = 5$**

remains stuck at 0, indicating that the disguise network has been utterly unable to transform negative data into samples the discriminator would classify as positive.

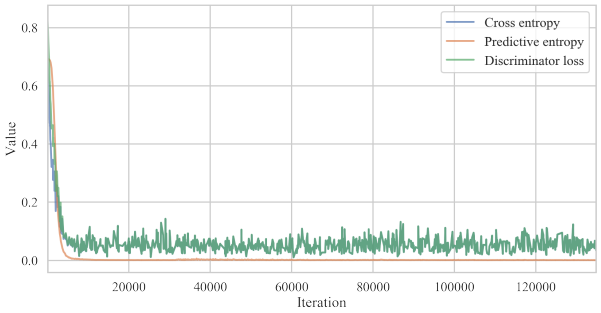

Figure 16: **Discriminator network metrics at $\lambda = 5$**

Because the predictive entropy is a measure of the discriminator network's prediction uncertainty for disguised samples, the collapse of the disguise quality towards extreme negative values necessarily results in the collapse of the predictive entropy towards 0. This causes the discriminator loss to become equivalent to the discriminator network's cross entropy (Figure 16) since the discriminator loss is a weighted sum of cross entropy and predictive entropy. Assuming that the disguise network is fully responsible for the predictive entropy approaching 0, a collapsed disguise network would serve no useful purpose since the discriminator would train as if the disguise network did not exist. In actuality, the predictive entropy could also approach 0 because it was successfully minimized by the discriminator network.

Setting λ to a low value grants the disguise network more freedom to drastically alter incoming negative samples, but at too low a value, the output of the disguise network may collapse towards the positive class. That is, if the disguise network is unrestricted

in how it may transform negative samples, its output will collapse to the same disguised sample, one that has a high probability of being classified as positive by the discriminator network.

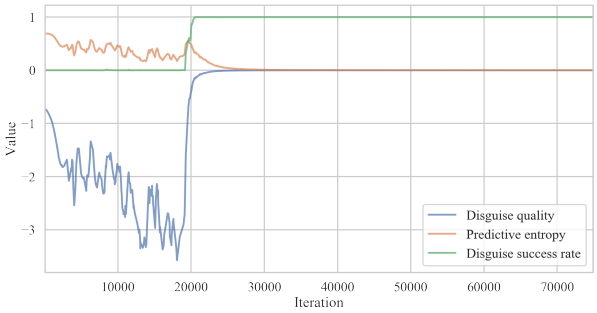

Figure 17: **Disguise network metrics at $\lambda = 0.5$**

Figure 17 shows how the disguise quality, disguise success rate, and predictive entropy unfold as a DAN trains with $\lambda = 0.5$. The disguise quality term collapses towards 0, meaning all disguised samples are confidently predicted by the discriminator to be positive.

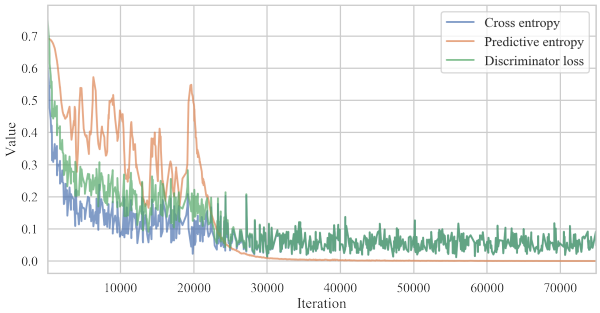

Figure 18: **Discriminator network metrics at $\lambda = 0.5$**

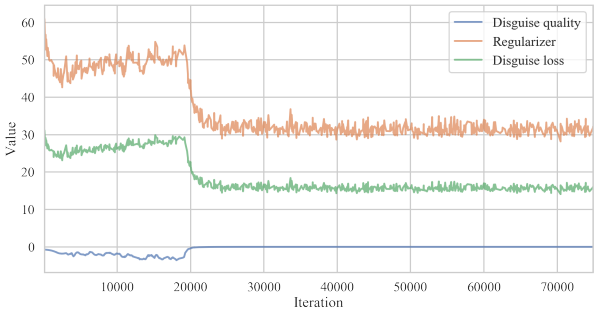

Figure 19: **Disguise loss at $\lambda = 0.5$**

We observed a surprising result in the relationship between disguise quality and the regularization term when the DAN trains at $\lambda = 0.5$; we normally expect increases in the disguise quality to coincide with increases in the regularization term, but at around iteration 20000, the collapse of the disguise quality towards 0 coincides with a significant drop in the regularization term. A possible explanation is that if the disguise network is able to learn that only a few features are relevant to successfully disguising negative samples, then it will also learn to perform identity transformations on all non-relevant features to minimize the regularization term.

Figure 20 shows the scaled transformation magnitudes (absolute value difference, normalized across all differences) of 400 negative CPC samples embedded into a 340-dimensional feature space. Some features consistently undergo drastic transformations while others remain untouched, supporting our theory for why an increase in disguise quality can accompany a simultaneous decrease in the regularization term.

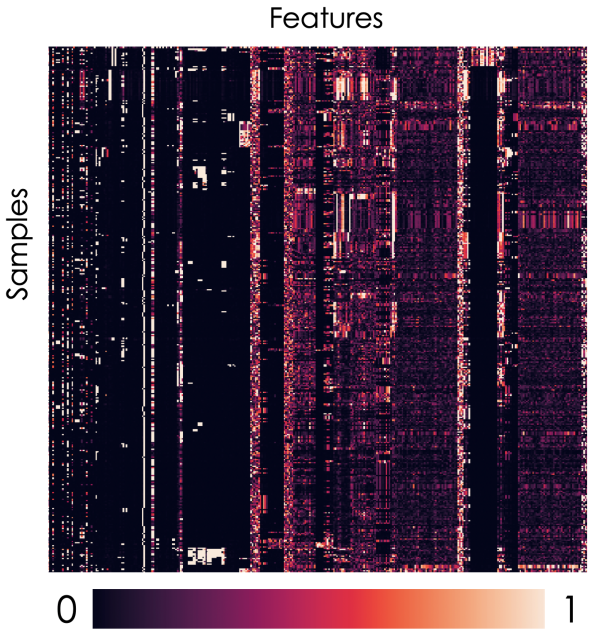

Figure 20: **Heat map of scaled transformation magnitudes on a sample of embedded CPC data**

To confirm our intuitions about how different settings of λ should affect the disguise network, we projected the CPC data and its disguises down to two dimensions via PCA (Figure 21), accounting for approximately 25% of the data variability. As with the plots in Figure 14, we see that higher λ results in more diverse but less successful transformations.

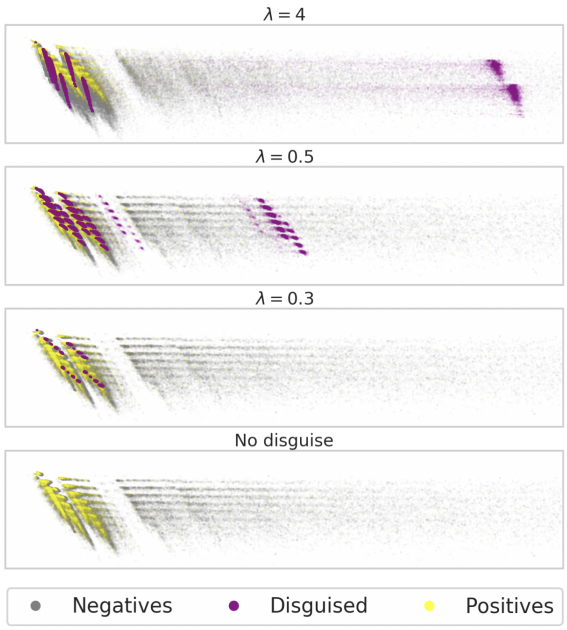

Figure 21: **Effect of λ on disguises of CPC data**

7.2 Possible inconsistencies with original implementation

Through unit testing, metric logging, experimentation on synthetic data, and visualization of the DAN's effects on MediaMath data, we have ensured the correctness of our DAN implementation to the best of our understanding. We reached out to Deng et al. multiple times with clarifying questions surrounding the implementation and training of DANs, but received no response, so proceeded with development and testing under our own assumptions. Below are some of the questions we sought answers to:

- **What hyperparameters were used to produce the published results on the benchmark datasets?** Although we conducted our own hyperparameter search, knowledge of the most effective hyperparameter settings Deng et al. found for their datasets would have 1) given us a better starting point for our search and 2) helped us understand better the relationship between λ and η.
- **Why is the first term of the disguise objective function considered a KL divergence between the predicted probability of positive classification and the positive data probability density function?** Deng et al. make this interpretation of the term we have dubbed the negative of the "disguise quality." We do not believe this is correct given the formulation of the disguise objective function and asked for clarification. If there indeed is a mismatch between formulation and interpretation, did the authors misinterpret the formulation, or did they mis-formulate their intention? We were interested in this clarification because the GAN objective function given by Deng et al. was not correct. Absent a response from the authors, we implemented the disguise objective function as it was formulated rather than its interpretation since the former encouraged reasonable behavior from the disguise network.
- **Was it necessary to use different learning rates for the disguise and discriminator networks?** We assumed this to not be the case, though would not be surprised if it were. We observed from the embedded CPC data that even with a 340-dimensional feature space and a disguise network that bottlenecks at a 256-node layer, the disguise network struggles to learn the identity function. Setting a higher value of λ encourages the disguise network to behave more like an autoencoder, however because the gradients used to update the network would be much larger, the disguise network would struggle to converge. Setting a different learning rate for optimizing the disguise network would allow λ to be arbitrarily high without affecting convergence of the discriminator network.
- **Did either the disguise or discriminator network require some form of pre-training?** Allowing the disguise network to train exclusively for some duration could aid in preventing it from undergoing positive collapse. This seems especially useful given the troubles we have seen in encouraging the disguise network to behave like an autoencoder while training near-simultaneously with the discriminator network.

7.3 Generalizing to substantially different datasets

Were the tuning and consistent implementation of the DAN not an issue, it remains possible that the benefits gained from applying the DAN on Deng et al.'s benchmark datasets do not translate to MediaMath's datasets. A key difference between the benchmark datasets and MediaMath's datasets is that while the benchmark data's dimensionality following one-hot encoding is around 200, MediaMath's data would expand to well over 100000 dimensions following the

same procedure. This means that the disguise network would need to be significantly larger to have the capacity to learn an identity on data in excess of 100000 dimensions, yet given our implementation of the DAN that does not use sparse matrices, a disguise network of such size would have been unfeasible to test.

The embedding pipeline and EmbeddingTransformer were built to address the challenges posed by high-cardinality data, but because Deng et al. had no need for an embedded representation of their benchmark data, the introduction of such transformations could very well render our DAN implementation incomparable to theirs. One possible issue is that a DAN trained on embeddings may never converge unless the embeddings are pre-trained. The embeddings are updated through the discriminator network and its corresponding loss function, which itself is evaluated in part using the output of the disguise network. The disguise network takes embeddings as input, so a scenario could exist where the disguise network causes an update in the embeddings, forcing it to then learn on an altered input, which in turn causes further updates to the embeddings. In other words, asking the disguise network to learn on embedded representations that have not been pre-trained may force the disguise network to train to an ever-shifting target.

7.4 Strategic errors

Because we cannot rule out the previous three issues as reasons for the DAN's underwhelming performance, we are prevented from concluding with reasonable certainty that the DAN is theoretically unsound and does not work. This lingering uncertainty regarding the DAN's utility is due in part to a strategic error we made in the early stages of this project.

Deng et al. evaluated the DAN on their benchmark datasets by using a sliding time series split that is 1) cumbersome to implement and 2) does not apply to the MediaMath data. We opted to forego reproducing this sliding time series split on the benchmark data which meant any results we generated from the original datasets would not be comparable to Deng et al.'s published results. For this reason, we proceeded directly with tuning and evaluating the DAN on MediaMath datasets.

In hindsight, a more prudent course of action would have been to at least evaluate our DAN implementation on Deng et al.'s benchmark datasets with a standard train-validation-test split. Applying such a testing framework to the benchmark datasets would not have produced results comparable to those already published, but the results would have controlled for the idiosyncrasies inherent to different datasets. Had we performed this step, we would have been able to make stronger statements about the general soundness of the DAN. That we did not was a case of allowing the perfect be the enemy of the good. As a result, testing this DAN implementation on the original benchmark datasets remains a highly worthwhile avenue of investigation.

8 Conclusion

To address the problem of classification in MediaMath's datasets with highly imbalanced class distributions, we implemented a Disguise Adversarial Network, initially proposed by Deng et al. It can be broadly understood as a minority class augmentation technique enabled by the disguise network combined with semi-supervised learning enabled by the discriminator network. We initially tested the model on 2D simulated data in order to fully understand how it transformed the data and verify that the model was working properly. We then created pipelines for processing two MediaMath datasets (CPC and CTR) with both one-hot encodings and categorical embeddings, and evaluated the model on the two datasets. After an extensive hyperparameter search, the model performed roughly as well on these datasets as a simple logistic regression. Once the data was artificially skewed, the DAN then outperformed the logistic regression, but still did not surpass the performance of a logistic regression using positive class oversampling. Testing the model on either the CPC or CTR dataset led to equivalently unremarkable results.

Given the differences between the MediaMath and original DAN datasets, we cannot be certain whether DANs are generally useful for most imbalanced datasets. However, it appears that for datasets of high dimensionality and good linear separability, the DAN does not improve upon already established oversampling and undersampling techniques, most of which often train faster and tend to be more adaptable. This seems to be true whether the categorical features of the data are one-hot encoded or fed through an embedding pipeline, as well as across different sizes and types of disguise and discriminator networks. Furthermore, DANs, much like GANs, require an involved training and tuning process. Though the results initially seemed promising,

its lack of success over other more established methods and its finicky nature mean that it can not be recommended for widespread industry use without further research.

References

[1] Nitesh V. Chawla et al. "Smote: synthetic minority over-sampling technique". In: *JAIR* 16 (2002), pp. 321–357.

[2] Yue Deng, Yilin Shen, and Hongxia Jin. "Disguise Adversarial Networks for Click-through Rate Prediction". In: *IJCAI* (2017), pp. 1589–1595.

[3] Haibo He et al. "Adasyn: Adaptive synthetic sampling approach for imbalanced learning". In: *IJCNN* (2008), pp. 1322–1328.