Disguise Adversarial Networks for Imbalanced Ad Conversion Datasets

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

1 Abstract

A common issue in using machine learning to predict ad conversions in click-through rate (CTR) 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 CTR datasets where the positive class typically comprises between 0.1% to 1% of the training data. MediaMath wishes to investigate the effectiveness of an experimental architecture for addressing class imbalance: the Disguise Adversarial Network (DAN).

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
- **Adaptive Synthetic Sampling Approach (ADASYN)**: a variation on SMOTE that adds Gaussian noise to the synthetic positive sample

Researchers at Samsung Research America (Deng et al.) recently proposed a novel architecture for handling class imbalance. Drawing inspiration from the surprising effectiveness of Generative Adversarial Networks (GAN's), 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, DAN's "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. Rather, an upsampling 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.

The DAN consists of two densely connected networks (Figure 1). The discriminator network D is the

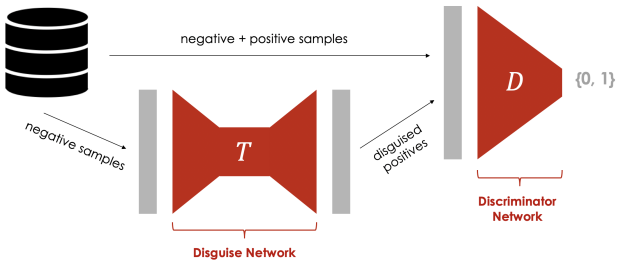

Figure 1: **DAN architecture**

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 $\mathcal{L}_1$:

$$\text{minimize } \mathcal{L}_1(T) = \underbrace{-\mathbb{E}_{x^- \sim P_{\Omega^-}}\left[\log D\left(T(x^-)\right)\right]}_{\text{Want } D(T(x^-))=1} + \underbrace{\lambda \|T(x^-) - x^-\|_1}_{\text{Want } T(x^-) \approx x^-}$$

Disguise Quality assesses how well T transforms negative samples to look like positive samples

Regularizer restricts T from making too drastic of transformations to negative samples

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

$$\begin{aligned} \text{minimize } \mathcal{L}_2(D) = &-\mathbb{E}_{x^- \sim P_{\Omega^-}}\left[\log\left(1 - D(x^-)\right)\right] && \text{Want } D(x^-) = 0 \\ &-\mathbb{E}_{x^+ \sim P_{\Omega^+}}\left[\log\left(D(x^+)\right)\right] && \text{Want } D(x^+) = 1 \end{aligned}$$

Figure 3: **A naive discriminator loss**

Unfortunately, this naive 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.

$$\begin{aligned} \text{minimize } \mathcal{L}_2(D) = &-\mathbb{E}_{x^- \sim P_{\Omega^-}}\left[\log\left(1 - D(x^-)\right)\right] && \text{Want } D(x^-) = 0 \\ &-\mathbb{E}_{x^+ \sim P_{\Omega^+}}\left[\log\left(D(x^+)\right)\right] && \text{Want } D(x^+) = 1 \\ &+\eta \mathbb{E}_{x^- \sim P_{\Omega^-}}\left[H\left(D\left(T(x^-)\right)\right)\right] && \text{Want } D(T(x^-)) \approx \text{0 or 1 but not 0.5} \end{aligned}$$

where $H(\hat{y}) = \underbrace{-[\hat{y}\log(\hat{y}) + (1-\hat{y})\log(1-\hat{y})]}_{\text{Entropy of Predictive Distribution}}$

measures the degree of uncertainty in D's predictions on disguised samples

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. 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

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

	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**

	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**

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.

An especially difficult aspect of both datasets is the extremely high cardinality of many of their categorical features, some of which are reported in Figure 2. This precludes the possibility of one-hot encoding, as such a transformation would prohibitively explode the dimensionality and physical size of the data. The team is currently considering two methods for addressing this issue.

- **Feature hashing**: a randomized binning; essentially a non-invertible many-to-less mapping of a set of categorical values to a more manageable, if less interpretable, set of categorical values
- **Learned embeddings**: training a low-dimensional dense representation of each categorical value, not unlike Word2Vec or GloVe

5 Evaluation plans

While we initially discussed attempting to reproduce Deng et al.'s results on publicly available datasets, the means by which they evaluate the DAN on these datasets varies significantly from how evaluations would be performed on MediaMath data. The authors argue that because some ads are only relevant for a short period of time, it makes sense to train and evaluate models in a manner that accounts for temporal effects. To this end, they do the following:

- Sort the dataset chronologically
- Divide the sorted dataset into 100 chunks
- Train and evaluate a model using a sliding window of 25 chunks, where the first 20 chunks form the training data and the last 5 chunks form the testing data

An attempt to reproduce the paper's results exactly would necessitate implementing an evaluation framework not applicable to the MediaMath datasets. For this reason, we chose to focus on evaluating the DAN with only MediaMath data. We believe that if we can show DANs outperform baseline models, positive class upsampling, SMOTE, and ADASYN, then this is sufficient confirmation of Deng et al.'s claims.

In the best case scenario, the DAN would outperform existing MediaMath benchmarks on these datasets. Less ideal but still encouraging is the mixed-case scenario, where the DAN exceeds our own baselines, but still does not perform as well as those of MediaMath's. In this situation, the mismatch in performance between our baselines and MediaMath's benchmarks would likely be attributable to differences in hyperparameter tuning and data processing, which once adjusted for, should allow the DAN to perform better. In the worst case, the DAN is not able to outperform our own baselines, in which case we would need to consider further modifications to the architecture or backtracking to the publicly available datasets.

6 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 5 illustrates how the core components of the DAN library interact.

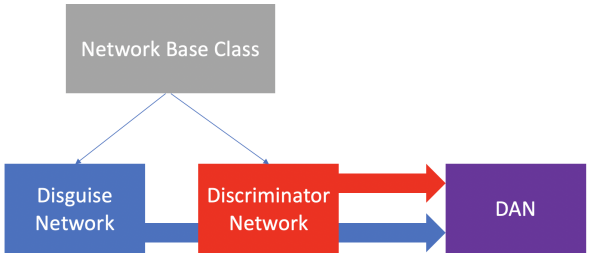

Figure 5: **An object-oriented implementation of the DAN**

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 6.

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

Figure 6: **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 will measure baseline performance.

As of writing, the DAN class exposes only three methods to the user:

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

Currently, these methods only support reading NumPy arrays fully loaded into memory. One of our immediate goals will be to adapt them to also accept data generators as arguments.

7 Results on synthetic data

As an initial proof of concept while waiting for the MediaMath datasets to be processed into a model-ingestible form, 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 is useful for unit testing and visualizing the behavior of the DAN. Figure 7 illustrates the distribution of data for the two classes.

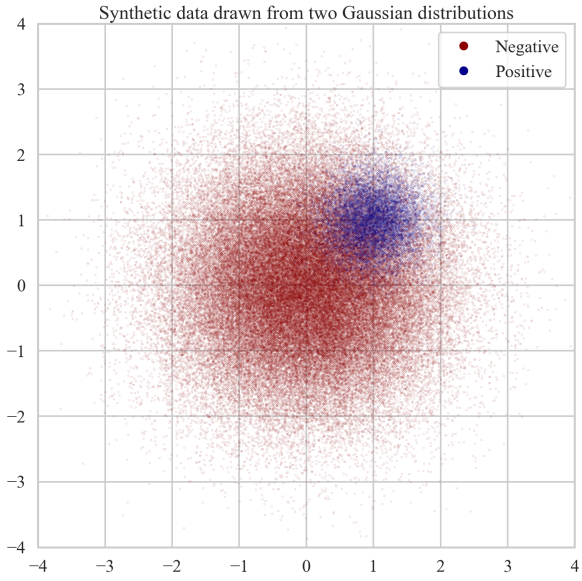

Figure 7: **Synthetic data**

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 8 shows the distribution of 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.

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

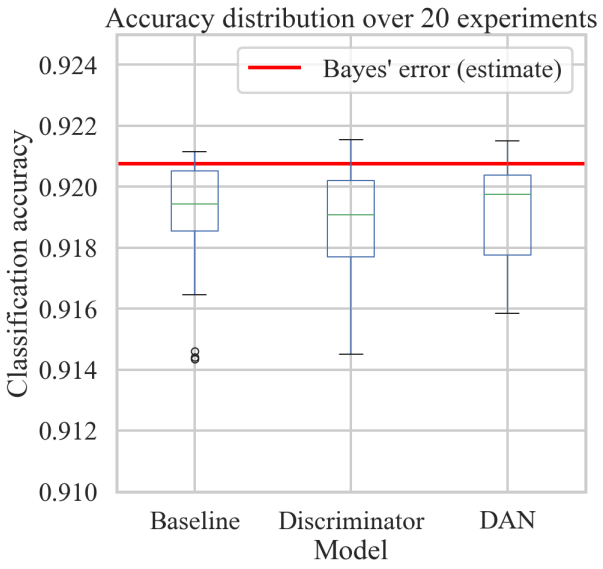

Figure 8: **Validation accuracy on synthetic data**

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 9).

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.

8 Next steps

Having developed an initial implementation of the DAN library, explored its behavior on synthetic data, and tested its components for general correctness, we are currently working towards the following set of goals:

- Building a preprocessing pipeline that will transform the datasets into a model ingestible form. These should consist of command-line executable scripts that will impute and scale the data as well as re-encode categorical features, either through one-hot encoding, feature hashing then one-hot encoding, or label encoding for training embedded representations.
- Developing **fit_generator()**, **predict_generator()**, and **transform_generator()** methods for the DAN class so that entire datasets do not need to be read into memory at once.
- Enabling the DAN's fit-related methods to accept validation sets so users can check for overfitting as the model trains.
- Incorporating multiple embedding layers into the DAN's architecture. This modification is uncharted territory because Deng et al.'s architecture does not account for the need to use embedded representations of high-cardinality feaures.
- Writing scripts for applying classical imbalanced data processing techniques such as SMOTE and ADASYN to the MediaMath datasets.
- Designing an experimental framework for automating the training, evaluation, logging, and visualization of models and their results.

Although we have shown rapid progress in model prototyping and development, organizationally, we are currently bottlenecked at building our data transformation pipeline. We hope that the pace of progress on this front will improve so that we may quickly move on to the next phases of the project.

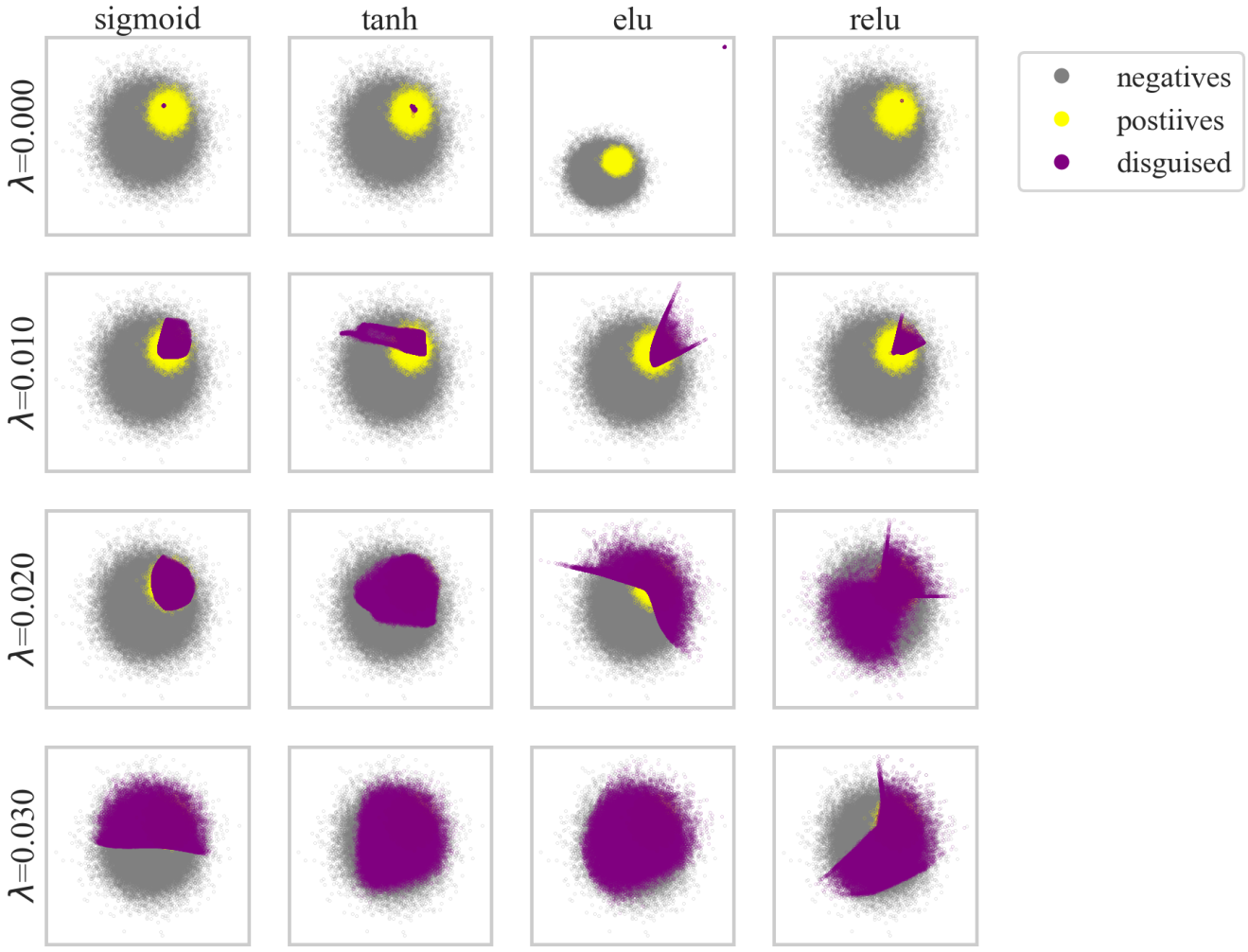

Figure 9: **Disguise transformations by activation and λ**