



Optimizing non-decomposable measures with deep networks

Amartya Sanyal^{1,2} · Pawan Kumar³ · Purushottam Kar³ · Sanjay Chawla⁴ · Fabrizio Sebastiani⁵

Received: 28 January 2018 / Accepted: 18 June 2018 / Published online: 2 July 2018
© The Author(s) 2018

Abstract

We present a class of algorithms capable of directly training deep neural networks with respect to popular families of task-specific performance measures for binary classification such as the F-measure, QMean and the Kullback–Leibler divergence that are structured and non-decomposable. Our goal is to address tasks such as label-imbalanced learning and quantification. Our techniques present a departure from standard deep learning techniques that typically use squared or cross-entropy loss functions (that are decomposable) to train neural networks. We demonstrate that directly training with task-specific loss functions yields faster and more stable convergence across problems and datasets. Our proposed algorithms and implementations offer several advantages including (i) the use of fewer training samples to achieve a desired level of convergence, (ii) a substantial reduction in training time, (iii) a seamless integration of our implementation into existing symbolic gradient frameworks, and (iv) assurance of convergence to first order stationary points. It is noteworthy that the algorithms achieve this, especially point (iv), despite being asked to optimize complex objective functions. We implement our techniques on a variety of deep architectures including multi-layer perceptrons and recurrent neural networks and show that on a variety of benchmark and real data sets, our algorithms outperform traditional approaches to training deep networks, as well as popular techniques used to handle label imbalance.

Keywords Optimization · Deep learning · F-measure · Task-specific training

Editors: Jesse Davis, Elisa Fromont, Derek Greene, and Bjorn Bringmann.

✉ Amartya Sanyal
amartya18x@gmail.com

✉ Pawan Kumar
kpawan@cse.iitk.ac.in

✉ Purushottam Kar
purushot@cse.iitk.ac.in

Extended author information available on the last page of the article

1 Introduction

As deep learning penetrates more and more application areas, there is a natural demand to adapt deep learning techniques to area and task-specific requirements and constraints. An immediate consequence of this is the expectation to perform well with respect to task-specific performance measures. However, this can be challenging, as these performance measures can be quite complex in their structure and be motivated by legacy, rather than algorithmic convenience. Examples include the F-measure that is popular in retrieval tasks, various ranking performance measures such as area-under-the-ROC-curve, and the Kullback–Leibler divergence that is popular in class-ratio estimation problems.

Optimizing these performance measures across application areas has proved to be challenging even when learning linear models, as is evidenced by the recent surge in methods for optimizing “non-decomposable” loss functions for learning linear models, as we review in Sect. 2. The challenge becomes doubly hard when trying to do so while training neural network architectures such as multi-layer perceptrons and convolutional or recurrent neural networks.

The vast majority of contemporary training techniques for neural networks consist of using simple per-sample loss functions such as least-squares loss or cross-entropy. While their use has allowed research directions to focus more on developing more evolved network architectures, as well as developing highly optimized implementations of training routines on GPU architectures, we show that this is suboptimal and that a sound effort towards training with task-specific loss functions pays off handsomely.

In particular, we will show how to train neural networks to directly optimize (smoothed versions of) non-decomposable performance measures such as F-measure, Q-mean and Kullback–Leibler divergence which are popularly used in binary classification with label imbalance and quantification (or class ratio estimation) settings. The problem of quantification expects accurate estimation of relative prevalence of class labels (e.g., fraction of positive vs negative reviews for some product or some campaign) and is useful in social engineering and epidemiology. We will apply our techniques to label imbalanced learning problems on several benchmark datasets as well as a sentiment analysis challenge task on Twitter feeds. We will find that in general, our proposed methods offer superior performance, while utilizing fewer samples and training time.

2 Related work

The recent years have seen much interest, as well as progress, in training directly with task-specific performance measures in the field of classification and ranking.

2.1 Works on non-decomposable measures with linear models

1. Koyejo et al. (2014) and Narasimhan et al. (2014) use *plug-in* classifiers for optimizing non-decomposable performance measures for binary classification such as F-measure and Q-mean. These methods first learn a class probability prediction model (typically using logistic regression) and then finely tune a threshold to get a classifier.
2. Kar et al. (2013, 2016), Narasimhan and Agarwal (2013b) and Narasimhan et al. (2015) use stochastic gradient descent-style algorithms for optimizing non-decomposable performance measures for binary classification and quantification problems such as F-measure, Q-mean, KL-divergence etc.

3. Kar et al. (2014, 2015) and Narasimhan and Agarwal (2013a,b) focus on optimizing ranking loss functions such as (partial) area under the ROC curve (pAUC), and precision at the top (prec@k).

However, all the above works focus only on training linear models. Although this allows for simple algorithms and detailed analyses and theoretical guarantees, the approaches do not directly extend to deep networks (we will discuss the challenges shortly). Algorithms for deep learning which directly optimize non-decomposable performance measures are relatively less explored.

This can be attributed to the use of back-propagation style algorithms for training neural networks which require calculating model gradients with respect to a batch of training points. With decomposable loss functions, where it is possible to define the gradient of the performance measure with respect to a single data point, this can be cheaply done – the batch gradient is simply the mean of gradients with respect to individual data points in that batch. However, the same becomes infeasible for non-decomposable performance measures (for example, it is meaningless to talk about the F-measure of a classifier on a single data point).

2.2 Works on non-decomposable measures with deep models

We are aware of a few efforts at training deep networks with non-decomposable losses. Most of these works focus on ranking tasks.

- (1) Song et al. (2016) train neural networks for ranking tasks with mean average precision (MAP) as the performance measure, by showing that for nicely behaved ranking loss functions, the expected gradient of the loss function with respect to the network weights can be expressed in terms of loss functions such as least squares loss.
- (2) Eban et al. (2017) train neural networks for ranking objectives e.g. area under the precision-recall curve and precision at a fixed recall rate.
- (3) Schäfer and Hüllermeier (2018) train neural networks for label ranking tasks based on the Plackett–Luce model where label features are available.

3 Our contributions

Our work advances the state of the art in training neural networks on a wide variety of non-decomposable performance measures for classification and quantification tasks. This complements existing works on ranking tasks.

- (1) We show how to train neural networks directly using (smoothed versions of) performance measures that are concave, pseudo-linear, or nested concave functions including F-measure, Q-mean, KL-divergence and many others.
- (2) We offer formal stabilization guarantees for all our algorithms, i.e., we guarantee that our algorithms rapidly approach a stable point where the effective gradients of the algorithm vanish.
- (3) Our methods offer far superior performance than traditional cross-entropy based training routines—on an F-measure maximization task on a benchmark dataset a9a, our method achieves an F-measure of around 0.68 in less than 10 mini-batch iterations whereas it takes traditional cross-entropy based training more than 80 iterations to reach similar performance levels.
- (4) Our methods also outperform popular techniques used to deal with label imbalance such as cost-weighted classification, plug-in methods, and loss-augmented inference methods

using structural surrogates—on a min-TPR/TNR maximization task on a benchmark dataset IJCNN, loss-augmented inference is only able to offer a performance level of 0.55 whereas our technique is able to reach performance over 0.95 in very few iterations.

- (5) We apply our techniques to a Twitter sentiment analysis challenge task using an end-to-end recurrent neural network with attention model and achieve near perfect quantification scores using a substantially smaller number of training iterations as compared to standard cross-entropy based training.

Our algorithms are readily adapted to neural architectures such as multi-layered perceptrons and recurrent networks, as well as easily integrated into popular symbolic gradient frameworks such as TensorFlow, PyTorch, and Theano.

3.1 Our contributions in the context of related work in deep models

- (1) Song et al. (2016), Eban et al. (2017) and Schäfer and Hüllermeier (2018) are focused on ranking measures whereas our work addresses binary classification and quantification (class-ratio estimation) measures.
- (2) Song et al. (2016) uses loss-augmented inference techniques (Joachims et al. 2009; Tsochantaridis et al. 2005) which are very well established in machine learning. We apply the same techniques to our performance measures (see the STRUCT-ANN benchmark in our experiments) and show that our precise primal dual techniques far outperform methods based on loss-augmented inference.
- (3) Although Eban et al. (2017) does consider the F-measure which we also study, they do not report any experimentation with F-measure. A possible reason for this might be that their algorithm requires a constrained optimization problem to be solved that is challenging over deep networks. Our methods generalize to a large number of widely used measures, e.g., H-mean, G-mean, Jaccard coefficient, Q-measure, etc. which Eban et al. (2017) do not claim to handle.
- (4) Song et al. (2016), Eban et al. (2017) and Schäfer and Hüllermeier (2018) do not offer any convergence guarantees for their proposed algorithms whereas we do offer stabilization guarantees for our methods.

3.2 Our contributions in the context of related work in linear models

We note that our methods do adapt techniques that were earlier proposed for training linear models, such as in Narasimhan et al. (2015). However, we stress that this adaptation is neither trivial, nor immediate, and constitutes an independent contribution.

- (1) Previous works, such as Narasimhan et al. (2015) only consider linear models which lead to convex problems. A naïve and direct application of existing techniques to deep networks yields poor experimental results. For example, we could only obtain good accuracies on the challenging F-measure after we adapted existing methods to carefully use pretraining techniques.
- (2) Previous works consider only with convex surrogate functions since they desire the overall training problem to remain convex. We found this to give poor results with deep neural networks. We found much superior performance upon the use of (non-convex) surrogates such as the sigmoidal.
- (3) Despite working with non-linear models such as deep networks, and using non-convex surrogates such as the sigmoid, we are still able to provide stabilization guarantees for

Table 1 List of performance measures $\Psi(P, N)$. We use the shorthand notation $P \equiv \text{TPR}$, $N \equiv \text{TNR}$. p, n denote the proportion of positives and negatives in the training dataset

Name	Type	Expression (P, N)
Min (Vincent 1994)	Concave	$\min\{P, N\}$
Q-Mean (Kennedy et al. 2010)	Concave	$1 - \sqrt{\frac{(1-P)^2 + (1-N)^2}{2}}$
F_β (Manning et al. 2008)	Pseudo-linear	$\frac{(1+\beta^2) \cdot P}{\beta^2 + n/p + P - n/p \cdot N}$
KLD (Barranquero et al. 2015)	Nested concave	See text

our algorithms. In fact, for F-measure and other pseudo-linear measures, our methods converge to a stationary point of the (smoothed) performance measure. Existing theoretical analyses on linear methods rely on convexity and cannot give such results.

4 Problem setting

We will consider binary classification and quantification problems in this work. Let $\mathcal{X} \subset \mathbb{R}^d$ be the space of feature vectors and $\mathcal{Y} = \{-1, +1\}$ be the label set. The training data set $S = (\mathbf{x}_i, y_i)_{i=1, \dots, n}$ shall be sampled i.i.d. from some fixed but unknown distribution \mathcal{D} over $\mathcal{X} \times \mathcal{Y}$. The proportion of positives in the population and sample S will be denoted by $p = \mathbb{P}_{(\mathbf{x}, y) \sim \mathcal{D}} [y = +1]$ and \hat{p}_S respectively.

In sharp contrast to most previous works on non-decomposable performance measures that consider only linear models, we concentrate on *non-linear models*, especially those induced by neural networks. Let us fix the neural architecture (layers, activations, connectivity) and let \mathcal{W} denote the space of all models (weights on the network edges). We will train a neural model, whose edge weights are indexed by $\mathbf{w} \in \mathcal{W}$, to assign a *score* to every data point $\mathbf{x} \in \mathcal{X}$ (that can be converted into a binary label by thresholding, to class probability estimates by applying a transfer function, etc). Linear models assign a score by simply computing $\langle \mathbf{w}, \mathbf{x} \rangle$.

We will use a more general notation $f(\mathbf{x}; \mathbf{w})$ (or equivalently $f_{\mathbf{w}}(\mathbf{x})$ as a shorthand) to denote the score given to the data point \mathbf{x} by the neural model indexed by the weights \mathbf{w} . The function f can be seen as encoding all the neural connections and activations. We stress that the function f is neither convex nor concave. This lack of structure in the scoring function precludes a large body of work in linear multivariate optimization and quantification from being applied to deep models.

Performance measures We consider performance measures that can be expressed in terms of the true positive rate (TPR—fraction of positively labeled points correctly labeled as positive) and true negative rate (TNR—fraction of negatively labeled points correctly labeled as negative) of the model. We will consider three general classes of performance measures described below. In our experiments, we will present results on a selection of these performance measures listed in Table 1. Below we recall the definitions of these families from Kar et al. (2016) and Narasimhan et al. (2015).

Pseudo-linear performance measures These measures can be written as a ratio of two linear functions of the TPR and TNR values, i.e. they have a fractional linear link function. More specifically, given given coefficients $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$,

$$\mathcal{P}_{(\mathbf{a}, \mathbf{b})}(\mathbf{w}) = \frac{a_0 + a_1 \cdot \text{TPR}(\mathbf{w}) + a_2 \cdot \text{TNR}(\mathbf{w})}{b_0 + b_1 \cdot \text{TPR}(\mathbf{w}) + b_2 \cdot \text{TNR}(\mathbf{w})}$$

The popularly used F-measure (Manning et al. 2008) is actually a pseudo-linear performance measure in terms of the TPR, TNR values of a model although it is more commonly represented as the harmonic mean of precision and recall. Other members include the Jaccard coefficient and the Gower–Legendre measure.

Concave performance measures These measures can be written as a concave function of the TPR and TNR values:

$$\mathcal{P}_\Psi(\mathbf{w}) = \Psi(\text{TPR}(\mathbf{w}), \text{TNR}(\mathbf{w}))$$

for some concave *link/wrapper function* $\Psi : \mathbb{R}^2 \rightarrow \mathbb{R}$. These measures are popular in cost-sensitive classification in cases with severe label imbalance, for example detection theory (Vincent 1994). A commonly used member of this family is the so-called Min-function assigns the value $\min\{\text{TPR}(\mathbf{w}), \text{TNR}(\mathbf{w})\}$ to a model \mathbf{w} . Note that this compels the model to pay equal attention to both classes. Other examples include the Q-mean and H-mean measures.

Nested concave performance measures Recent works, e.g., Barranquero et al. (2015) and Kar et al. (2016) in areas such as quantification and class ratio estimation, have brought focus on performance measures that can be written as concave combinations of concave performance measures. More formally, given three concave functions $\Psi, \zeta_1, \zeta_2 : \mathbb{R}^2 \rightarrow \mathbb{R}$, we define a performance measure

$$\mathcal{P}_{(\Psi, \zeta_1, \zeta_2)}(\mathbf{w}) = \Psi(\zeta_1(\mathbf{w}), \zeta_2(\mathbf{w})),$$

where $\zeta_i(\mathbf{w}) := \zeta_i(\text{TPR}(\mathbf{w}), \text{TNR}(\mathbf{w}))$, $i = 1, 2$. A popular measure for quantification tasks is the *KLD: Kullback–Leibler Divergence* (Barranquero et al. 2015; Esuli and Sebastiani 2015; Gao and Sebastiani 2015; Kar et al. 2016) which can be shown to be a sum of concave functions of the TPR and TNR. If $\mathbf{p} \in \mathbb{R}^2$ is the vector of true class priors for a binary classification task and $\hat{\mathbf{p}}$ an estimate thereof, then

$$\text{KLD}(\mathbf{p}, \hat{\mathbf{p}}) = \sum_{y \in \mathcal{Y}} p(y) \log \frac{p(y)}{\hat{p}(y)} \tag{1}$$

$\text{KLD}(p, \hat{p}) = 0$ indicates perfect quantification.

There are several other performance measures that our techniques can handle but which we do not discuss here due to lack of space. These include measures for class-imbalanced classification such as H-mean, G-mean, Jaccard coefficient (Narasimhan et al. 2015), as well as quantification measures such as Q-measure, NSS and CQB (Kar et al. 2016).

Smoothing the performance measures Since TPR and TNR are count-based measures, they are unsuitable for numerical optimization algorithms. Thus, it is very common in learning and optimization literature to use *surrogate reward/loss functions* to stand in for them. Some approaches (Narasimhan et al. 2015) use point/decomposable surrogates whereas others (Narasimhan and Agarwal 2013a; Kar et al. 2015; Song et al. 2016) use non-decomposable surrogates.

We will use *point* surrogate reward functions to replace TPR, TNR in our objectives. A reward function r assigns a *reward* $r(\hat{y}, y)$ when the true label is $y \in \mathcal{Y}$ but the prediction is $\hat{y} \in \mathbb{R}$. Given a reward function $r : \mathbb{R} \times \mathcal{Y} \rightarrow \mathbb{R}$, a model $\mathbf{w} \in \mathcal{W}$, data point $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$, and scoring function f , we will use

$$\begin{aligned} r^+(\mathbf{w}; \mathbf{x}, y) &= \frac{1}{p} \cdot r(f(\mathbf{x}; \mathbf{w}), y) \cdot \mathbb{I}\{y = 1\} \\ r^-(\mathbf{w}; \mathbf{x}, y) &= \frac{1}{1-p} \cdot r(f(\mathbf{x}; \mathbf{w}), y) \cdot \mathbb{I}\{y = -1\}, \end{aligned}$$

to calculate rewards on positive and negative points ($\mathbb{I}\{\cdot\}$ denotes the indicator function). For purpose of training, we may replace p with \hat{p}_S i.e. the proportion of positives in the data sample.¹

We will use the shorthand $P(\mathbf{w}) = \mathbb{E}[r^+(\mathbf{w}; \mathbf{x}, y)]$ to denote population average of the reward function and, given a sample of n data points $S = (\mathbf{x}_i, y_i)_{i=1, \dots, n}$, denote the sample average as $\hat{P}_S(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n r^+(\mathbf{w}; \mathbf{x}_i, y_i)$ and similarly define $N(\mathbf{w})$, $\hat{N}_S(\mathbf{w})$. We will use $\hat{P}_S(\mathbf{w})$ and $\hat{N}_S(\mathbf{w})$ in place of TPR and TNR respectively while executing our algorithms.

Unlike previous work (Kar et al. 2016; Narasimhan et al. 2015), we will *not* restrict ourselves to concave surrogate reward functions. In particular we will utilize the *sigmoidal* reward, which is widely used as an activation function in neural networks is non-concave: $r_{\text{sigmoid}}(\hat{y}, y) = (1 + \exp(-y \cdot \hat{y}))^{-1}$. The reason for this is that the sigmoid resembles the misclassification loss function that lead to TPR and TNR, much more than the hinge loss or the logistic loss, and gave better performance. Many existing works preferred using losses such as the hinge and the logistic loss merely due to their convexity.

5 Deep optimization algorithms

The task of training deep models directly for non-decomposable performance measures is challenging due to several reasons: (1) these measures are non-decomposable and do not lend themselves to straightforward training methods such as gradient descent or back-propagation, (2) deep models do not offer the convenience of convexity, and (3) existing methods for optimizing such measures, e.g., Kar et al. (2016) and Narasimhan et al. (2015) fail to apply directly to deep models. We will see in Sect. 6 that a direct and naive application of traditional techniques yields poor results.

We will show how to overcome these challenges to arrive at scalable methods for training deep networks directly on (smoothed) non-decomposable measures. Let the procedure $\text{NN-init}(d_{\text{in}}, d_{\text{out}}, \text{conf})$ initialize a neural network with d_{in} input nodes, d_{out} output nodes, and internal configuration (hidden layers, number of internal nodes, connectivity) specified by conf . Please refer to Sect. 6 for details.

Intuition behind our Algorithms The basic intuition behind our algorithms is simple but useful, and one that has been used by previous works (Kar et al. 2016; Narasimhan et al. 2015) as well. We realize that training any model with respect to *decomposable* performance measures such as cross entropy or cost-weighted classification problems, is relatively simple, and can be accomplished using classical back-propagation. Thus, we convert the task of training with respect to a non-decomposable performance measure, say F-measure, into a sequence of tasks of training with respect to cost-weighted classification problems, with the costs being recalculated at each time step using methods specific to the performance measure at hand.

5.1 DAME: a deep learning technique for pseudo-linear performance measures

We present **DAME** (Algorithm 1), an algorithm for training deep models on pseudo-linear performance measures such as F-measure. We recall that although Eban et al. (2017) discuss F-measure optimization on deep models, they do not implement their algorithm or report

¹ Note that since $\mathbb{E}[r^+(\mathbf{w}; \mathbf{x}, y)] = \mathbb{E}[r(f(\mathbf{x}; \mathbf{w}), y) | y = 1]$, setting $r^{0-1}(\hat{y}, y) = \mathbb{I}\{y \cdot \hat{y} > 0\}$ i.e. classification accuracy as the reward function yields $\mathbb{E}[r^+(\mathbf{w}; \mathbf{x}, y)] = \text{TPR}(\mathbf{w})$.

experimental results with it, possibly since their method involves expensive constrained optimization problems involving deep networks. It will be our objective to maximize the following performance measure

$$\mathcal{P}_{(a,b),S}(\mathbf{w}) = \frac{a_0 + a_1 \cdot \hat{P}_S(\mathbf{w}) + a_2 \cdot \hat{N}_S(\mathbf{w})}{b_0 + b_1 \cdot \hat{P}_S(\mathbf{w}) + b_2 \cdot \hat{N}_S(\mathbf{w})},$$

where $\hat{P}_S(\mathbf{w})$, $\hat{N}_S(\mathbf{w})$ are the smoothed sample versions of TPR, TNR. Define $\mathcal{P}_{a,S}(\mathbf{w}) := a_0 + a_1 \cdot \hat{P}_S(\mathbf{w}) + a_2 \cdot \hat{N}_S(\mathbf{w})$ and $\mathcal{P}_{b,S}(\mathbf{w}) := b_0 + b_1 \cdot \hat{P}_S(\mathbf{w}) + b_2 \cdot \hat{N}_S(\mathbf{w})$. Our discussion will benefit from the notion of a *valuation function* from Narasimhan et al. (2015).

Definition 1 (*Valuation function*) Given coefficients $a_0, a_1, a_2, b_0, b_1, b_2$, and a level $v > 0$, define the valuation of a model at level v on training set S as

$$V_S(\mathbf{w}, v) = \mathcal{P}_{a,S}(\mathbf{w}) - v \cdot \mathcal{P}_{b,S}(\mathbf{w}) = (a_0 - vb_0) + (a_1 - vb_1) \cdot \hat{P}_S(\mathbf{w}) + (a_2 - vb_2) \cdot \hat{N}_S(\mathbf{w})$$

Intuition behind DAME Notice that a model \mathbf{w} has good performance, i.e., $\mathcal{P}_{(a,b)}(\mathbf{w}) > v$ for some $v > 0$ if and only if $\mathcal{P}_a(\mathbf{w}) \geq v \cdot \mathcal{P}_b(\mathbf{w})$ (F-measure and other performance measures always satisfy $\mathcal{P}_b(\mathbf{w}) > 0$). This is equivalent to $(a_0 - vb_0) + (a_1 - vb_1) \cdot \hat{P}_S(\mathbf{w}) + (a_2 - vb_2) \cdot \hat{N}_S(\mathbf{w}) \geq 0$ which is equivalent to $V_S(\mathbf{w}, v) \geq 0$, i.e., the model has a non-negative valuation at level v .

Thus, one way to maximize $\mathcal{P}_{(a,b)}$ is to simply check for various values of v if there exists any model \mathbf{w} such that $V_S(\mathbf{w}, v) \geq 0$ or not. The largest value v at which the answer is yes has to be the maximum achievable level of performance with respect to $\mathcal{P}_{(a,b),S}$ by the above argument. The work of Koyejo et al. (2014) does something similar. However, this may be very expensive since it is not clear which values v to query and how to find if the answer is yes or no.

DAME takes this intuition and turns it into a scalable algorithm. At every time step, it looks at the level $v^t = \mathcal{P}_{(a,b),S}(\mathbf{w}^t)$ since it is assured that at least one model, in particular \mathbf{w}^t , does satisfy $V_S(\mathbf{w}, v) \geq 0$. Then it queries if some other model \mathbf{w}' can beat \mathbf{w}^t in terms of valuation at level v^t . Note that the search for such as model \mathbf{w}' requires us to solve

$$\mathbf{w}' = \arg \max_{\mathbf{w} \in \mathcal{W}} V_S(\mathbf{w}, v^t) = \arg \max_{\mathbf{w} \in \mathcal{W}} (a_1 - v^t b_1) \cdot \hat{P}_S(\mathbf{w}) + (a_2 - v^t b_2) \cdot \hat{N}_S(\mathbf{w}),$$

which is simply a cost-weighted classification problem with the weights $(a_1 - v^t b_1)$ and $(a_2 - v^t b_2)$ on positives and negatives respectively. Having obtained an approximate maximizer of the above problem, say \mathbf{w}^{t+1} , **DAME** resets the level to $v^{t+1} = \mathcal{P}_{(a,b),S}(\mathbf{w}^{t+1})$ and challenges a model to maximize $V_S(\mathbf{w}, v^{t+1})$. Note that the costs would have changed now for this new challenge. However, this strategy, which was implemented by Narasimhan et al. (2015), performs poorly on deep networks. The complex nature of pseudo-linear performance measures, that are neither convex nor concave, seem to make it more challenging to train deep models by solving the intermediate cost-weighted problems effectively.

To overcome this, **DAME** first performs a pretraining step where the entire network is trained using cost-weighted classification losses using standard mini-batch SGD. Let $(\mathbf{w}_1, \mathbf{w}_2)$ denote a stacking of the neural networks described by the models \mathbf{w}_1 and \mathbf{w}_2 . More specifically \mathbf{w}_2 denotes a network with input dimensionality d_{in} and output dimensionality d_{int} whereas \mathbf{w}_1 denotes a network with input dimensionality d_{int} and output dimensionality d_{out} .

In the pretraining step, both $\mathbf{w}_1, \mathbf{w}_2$ are trained on a cost weighted classification problem where the cost on various classes is set naïvely to the inverse of their prevalence in the dataset, i.e., the cost of positives is $1/\hat{p}_S$. After this, a fine-tuning step is performed where *only the*

Algorithm 1 DAME: A Deep Alternating Maximization mEhod

Require: Training dataset $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, step lengths η_t , network configuration $\{d_{in}, d_{int}, d_{out}, \text{conf}_1, \text{conf}_2\}$, batch size B , proportion of positives \hat{p}_S

- 1: $\mathbf{w}_1^{-1,0} \leftarrow \text{NN-init}(d_{in}, 1, \text{conf}_1)$
- 2: $\mathbf{w}_2^{-1} \leftarrow \text{NN-init}(d_{in}, d_{int}, \text{conf}_2)$
- 3: $(\mathbf{w}_1^{0,0}, \mathbf{w}_2^0) \leftarrow$ Pre-train on weighted cross-entropy on dataset S
- 4: Create new dataset $\tilde{S} = \{(\mathbf{z}_i, y_i)\}_{i=1}^n$ where $\mathbf{z}_i = f(\mathbf{x}_i, \mathbf{w}_2^0)$ ▷ New features
- 5: **for** $t = 1, 2, \dots, T$ **do**
- 6: $S_{t,0} \leftarrow$ SAMPLE mini-batch of B data points $(\mathbf{z}_i^{t,0}, y_i^{t,0})$
- 7: $v^t \leftarrow \mathcal{P}(\mathbf{a}, \mathbf{b})_{S_{t,0}}(\mathbf{w}_1^{t-1,0}, \mathbf{w}_2^0)$
- 8: **for** $t' = 1, 2, \dots, T'$ **do**
- 9: $S_{t,t'} \leftarrow$ SAMPLE mini-batch of B data points $(\mathbf{z}_i^{t,t'}, y_i^{t,t'})$
- 10: $\mathbf{w}_1^{t-1,t'} \leftarrow \mathbf{w}_1^{t-1,t'-1} + \eta_{t'} \cdot \nabla_{\mathbf{w}_1^{t-1,t'-1}} V_{S_{t,t'}}((\mathbf{w}_1^{t-1,t'-1}, \mathbf{w}_2^0), v^t)$
- 11: **end for**
- 12: $\mathbf{w}_1^{t,0} \leftarrow \mathbf{w}_1^{t-1,T'}$
- 13: **end for**
- 14: **return** $(\mathbf{w}_1^{T,0}, \mathbf{w}_2^0)$

upper layers of the network, i.e., \mathbf{w}_1 are trained to optimize the F-measure. To save time during the fine-tuning phase, instead of optimizing the intermediate cost-weighted classification problems completely, DAME simply executes a few iterations of back-propagation with respect to the cost-weighted problems. This two-stage procedure offers much more performance than training with respect to valuation functions right from the start.

Assumptions for theoretical convergence results We are able to show a first order stationarity guarantee for DAME under the following assumptions:

- (1) The negated valuation functions are L -strongly smooth functions (see definition of a strongly smooth function below) of the upper model \mathbf{w}_1 . Since the sigmoidal reward function is $\frac{1}{2}$ -strongly smooth, this will be true for L that depends on the size of the network.
- (2) For some $M \geq m > 0$, the performance measure satisfies $\mathcal{P}_a(\mathbf{w}) \leq M$ and $\mathcal{P}_b(\mathbf{w}) \geq m$ for all $\mathbf{w} \in \mathcal{W}$. We note that these assumptions are standard (Kar et al. 2016; Narasimhan et al. 2015) and readily satisfied by the F-measure, Jaccard coefficient, etc., for which we have $m, M = \Theta(1)$ (see Narasimhan et al. 2015). We will denote $\kappa = 1 + M/m$.
- (3) The norm of the gradient of the reward function with respect to any single data point is bounded, i.e., $\max \{ \|\nabla_{\mathbf{w}} r^+(\mathbf{w}, \mathbf{x}, y)\|_2, \|\nabla_{\mathbf{w}} r^-(\mathbf{w}, \mathbf{x}, y)\|_2 \} \leq r$ for any model \mathbf{w} and any data point (\mathbf{x}, y) . This can be easily ensured by controlling the weights of the network and normalizing the data features \mathbf{x} .

Definition 2 (Strongly-smooth function) We call a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ strongly smooth if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, we have

$$f(\mathbf{x}) \leq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle + \frac{L}{2} \|\mathbf{x} - \mathbf{y}\|_2^2$$

Theorem 1 If executed with a uniform step length $\eta < \frac{2}{L\kappa}$ with a batch size B , DAME discovers an $\epsilon + \mathcal{O}\left(\frac{r}{m\sqrt{B}}\right)$ -stable model within $\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ inner iterations with high probability. More specifically, within $T \leq \frac{\kappa^2}{m} \frac{2}{\eta(2-L\kappa\eta)\epsilon^2}$ iterations, DAME identifies a model $\mathbf{w}_1^{T,0}$ such

that $\|\nabla_{\mathbf{w}} \mathcal{P}_{(\mathbf{a}, \mathbf{b})}(\mathbf{w}_1^{T,0})\|_2 \leq \epsilon + \mathcal{O}\left(\frac{r}{m\sqrt{B}}\right)$. If a mini-batch is not used and $S_{t,i} = \tilde{S}$ for all time steps t, i , then we are (deterministically) assured of an ϵ -stable model, i.e., that $\|\nabla_{\mathbf{w}} \mathcal{P}_{(\mathbf{a}, \mathbf{b})}(\mathbf{w}_1^{T,0})\|_2 \leq \epsilon$ instead.

We present the proof in [Appendix A](#). Since the pre-training phase enjoys local convergence guarantees by standard arguments, the argument is presented only for the fine-tuning phase and the lower network is omitted in the analysis.

5.2 DUPLÉ: a deep learning technique for concave performance measures

We present **DUPLÉ** (Algorithm 2), a scalable stochastic mini-batch primal dual algorithm for training deep models with concave performance measures. It will be our objective to maximize the following performance measure

$$\mathcal{P}_{\Psi, S}(\mathbf{w}) = \Psi(\hat{P}_S(\mathbf{w}), \hat{N}_S(\mathbf{w})),$$

where $\hat{P}_S(\mathbf{w}), \hat{N}_S(\mathbf{w})$ are smoothed sample versions of TPR, TNR and $\Psi : \mathbb{R}^2 \rightarrow \mathbb{R}$ is a concave link function. We shall find it convenient to define the (concave) Fenchel conjugate of Ψ in order to present the intuition behind our algorithms. For any concave function $\Psi : \mathbb{R}^2 \rightarrow \mathbb{R}$ and $\alpha, \beta \in \mathbb{R}$, define

$$\Psi^*(\alpha, \beta) = \inf_{u, v \in \mathbb{R}} \{\alpha u + \beta v - \Psi(u, v)\}. \tag{2}$$

Intuition behind DUPLÉ The reason behind defining the Fenchel conjugate is that it can be shown that for all concave, upper semi-continuous functions, by the Fenchel-Moreau theorem, we have, for any $u, v \in \mathbb{R}$,

$$\Psi(u, v) = \inf_{\alpha, \beta \in \mathbb{R}} \{\alpha u + \beta v - \Psi^*(\alpha, \beta)\}. \tag{3}$$

This reformulation is advantageous since an application of Danskin’s theorem now allows us to take (sub) gradients with respect to Ψ . More formally, Danskin’s theorem tells us that if $\tilde{\alpha}, \tilde{\beta}$ are the minimizer values in (3) at a point u^0, v^0 , then $\nabla_{u,v} \Psi(u, v)|_{(u^0, v^0)} \ni (\tilde{\alpha}, \tilde{\beta})$. This gives us a way to perform back-propagation-based training of neural networks with respect to concave performance measures: at every time step we can consider the current model \mathbf{w}^t , evaluate $\hat{P}_S(\mathbf{w}^t), \hat{N}_S(\mathbf{w}^t)$ using a forward pass over the entire training set S , solve the following problem

$$(\alpha', \beta') = \arg \inf_{\alpha, \beta \in \mathbb{R}} \{\alpha u + \beta v - \Psi^*(\alpha, \beta)\},$$

and then apply chain rule to obtain

$$\begin{aligned} \nabla_{\mathbf{w}} \Psi(\hat{P}_S(\mathbf{w}), \hat{N}_S(\mathbf{w}))|_{\mathbf{w}^t} &= \alpha' \cdot \nabla_{\mathbf{w}} \hat{P}_S(\mathbf{w})_{\mathbf{w}^t} + \beta' \cdot \nabla_{\mathbf{w}} \hat{N}_S(\mathbf{w})_{\mathbf{w}^t} \\ &= \nabla_{\mathbf{w}} (\alpha' \cdot \hat{P}_S(\mathbf{w})_{\mathbf{w}^t} + \beta' \cdot \hat{N}_S(\mathbf{w})_{\mathbf{w}^t}) \end{aligned}$$

Notice that this is identical to taking a gradient step with respect to a cost-weighted classification problem with cost α' on positives and cost β' on negatives. However, the process of discovering the optimal “dual variables” (α', β') this way is expensive since it requires a pass over the entire dataset S . **DUPLÉ** offers a much faster way to discover dual variables that, although not optimal w.r.t. \mathbf{w}^t , nevertheless promote progress. The advantage gained here is that we can find these variables using a single mini-batch rather than going over the entire dataset.

Algorithm 2 DUPLE: Dual UPdates for Learning dEep-models

```

Require: Primal step sizes  $\eta_t$ , network configuration  $\{d_{in}, \text{conf}\}$ , batch size  $b$ 
1:  $\mathbf{w}^0 \leftarrow \text{NN-init}(d_{in}, 1, \text{conf})$ 
2:  $\{\alpha^0, \beta^0, r_+, r_-, n_+, n_-\} \leftarrow \text{init}()$ 
3: for  $t = 1, 2, \dots, T$  do
4:    $S_t \leftarrow \text{SAMPLE mini-batch of } b \text{ data points } \{(\mathbf{x}_i^t, y_i^t)\}_{i=1, \dots, b}$ 
5:    $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} + \eta_t \cdot \nabla_{\mathbf{w}} g(\mathbf{w}^t; S_t, \alpha^{t-1}, \beta^{t-1})$  ▷ Primal Step
6:    $r_+ \leftarrow r_+ + \frac{1}{b} \sum_{i=1}^b r^+(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t)$  ▷ Tot. reward on +ves
7:    $r_- \leftarrow r_- + \frac{1}{b} \sum_{i=1}^b r^-(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t)$  ▷ Tot. reward on -ves
8:    $n_+ \leftarrow n_+ + \frac{1}{b} \sum_{i=1}^b \mathbb{I}\{y_i^t = +1\}$  ▷ Total # positives
9:    $n_- \leftarrow n_- + \frac{1}{b} \sum_{i=1}^b \mathbb{I}\{y_i^t = -1\}$  ▷ Total # negatives
10:   $(\alpha^t, \beta^t) \leftarrow \arg \min_{(\alpha, \beta)} \left[ \alpha \frac{r_+}{n_+} + \beta \frac{r_-}{n_-} - \Psi^*(\alpha, \beta) \right]$  ▷ Dual Step
11: end for
12: return  $\mathbf{w}^T$ 

```

The way **DUPLE** does this is by maintaining a proxy for the empirical reward values $\hat{P}_S(\mathbf{w}^t), \hat{N}_S(\mathbf{w}^t)$ such that this proxy is very cheap to update. At every time step, we have a model \mathbf{w}^t and a mini-batch S_t being used to train further. **DUPLE** stores the values $\hat{P}_{S_t}(\mathbf{w}^t), \hat{N}_{S_t}(\mathbf{w}^t)$ inside a running average. At any point, this running average is used as a cheap proxy for $\hat{P}_S(\mathbf{w}^t), \hat{N}_S(\mathbf{w}^t)$.

We note that **DUPLE** draws upon the **SPADE** algorithm proposed in Narasimhan et al. (2015). However, its application to deep models requires non-trivial extensions.

- (1) **SPADE** performs SGD steps with individual data points since gradient updates are rapid with linear models. Doing so with neural models is expensive.
- (2) The theoretical analysis of **SPADE** assumes that the scoring and reward functions being used are concave functions of \mathbf{w} . As noted in Sect. 4, for neural models, the scoring and reward functions are both non-concave.
- (3) Deep learning frameworks are highly optimized for gradient computations, assuming that the objective function with respect to which gradients are computed, is static across iterations (although this is not a drawback of some recent frameworks). **SPADE**, while applying the chain rule, ends up considering a different cost-weighted objective at each step.

DUPLE addresses the above issues and makes crucial design changes that make it highly optimized for use with deep networks.

- (1) **DUPLE** operates using cheap proxies that, at no point, require an pass over the entire dataset, or gradient computations on individual data points. Gradient computation is done only over mini batches which is a highly optimized operation in deep learning frameworks given that the batch sizes are not too large. We found this to also improve the stability properties of the algorithm.
- (2) At every time step, **DUPLE** requests gradients with respect to a cost-weighted objective function of the form $g(\mathbf{w}; S, \alpha, \beta) = \alpha \cdot \hat{P}_S(\mathbf{w}) + \beta \cdot \hat{N}_S(\mathbf{w})$. We exploit symbolic computation capabilities offered by frameworks such as Theano (Bergstra et al. 2010) to allow the scalars α, β to be updated dynamically and train the network efficiently on a different objective function at each time step.
- (3) Our analysis for **DUPLE** makes absolutely no assumptions on the convexity/concavity of the reward and scoring functions. It only requires both functions r^+, r^- to be differ-

entiable almost-everywhere. Thus, **DUPLE** only assumes the bare minimum to allow itself to take gradients.

We show the following convergence guarantee for **DUPLE** (see [Appendix B](#)) assuming that the negated surrogate reward function $-r(f(\mathbf{x}; \mathbf{w}), y)$ is L' -strongly smooth. This is satisfied by the sigmoidal reward function for some L' that depends on the size of the network. We nowhere assume the surrogate reward functions to be concave or convex. We will use the shorthand $\nabla^t = \nabla_{\mathbf{w}}g(\mathbf{w}^t; S_t, \alpha^t, \beta^t)$ and $F(\mathbf{w}^t, \alpha^t) = g(\mathbf{w}^t; S_t, \alpha^t, \beta^t)$. The following result assures us that the **DUPLE** procedure will stabilize rapidly and not oscillate indefinitely.

Theorem 2 *Consider a concave performance measure defined using a link function Ψ that is concave and L' -smooth. Then, if executed with a uniform step length satisfying $\eta < \frac{2}{L'}$, then **DUPLE** ϵ -stabilizes within $\tilde{\mathcal{O}}\left(\frac{1}{\epsilon^2}\right)$ iterations. More specifically, within T iterations, **DUPLE** identifies a model \mathbf{w}^t such that $\|\nabla^t\|_2 \leq \mathcal{O}\left(\sqrt{L' \frac{\log T}{T}}\right)$.*

5.3 DENIM: deep learning with nested concave performance measures

We extend the **DUPLE** algorithm to performance measures that involve a nesting of concave functions. To reiterate, the KLD performance measure which is used extensively for quantification, falls in this category. These measures are challenging to optimize using **DUPLE** due to their nested structure which prevents a closed form solution for the Fenchel conjugates.

Algorithm 3 DENIM: A DEep Nested primal-dual Method

Require: Primal step sizes η_t , network configuration $\{d_{\text{in}}, \text{conf}\}$, batch size b

- 1: $\mathbf{w}^0 \leftarrow \text{NN-init}(d_{\text{in}}, 1, \text{conf})$
 - 2: $\{\mathbf{r}^0, \mathbf{q}^0, \alpha^0, \beta^0, \gamma^0\} \leftarrow \text{init}()$
 - 3: **for** $t = 1, 2, \dots, T$ **do**
 - 4: $S_t \leftarrow \text{SAMPLE mini-batch of } b \text{ data points } \{\mathbf{x}_i^t, y_i^t\}_{i=1, \dots, b}$
 - 5: $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} + \eta_t \cdot \nabla_{\mathbf{w}}h(\mathbf{w}^t; S_t, \alpha^t, \beta^t, \gamma^t)$ ▷ Primal Step
 - 6: $\mathbf{q}^t \leftarrow (t-1) \cdot \mathbf{q}^{t-1} + (\alpha_1^{t-1}, \beta_1^{t-1}) \sum_{i=1}^b r^+(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t)$
 - 7: $\mathbf{q}^t \leftarrow \mathbf{q}^t + (\alpha_2^{t-1}, \beta_2^{t-1}) \sum_{i=1}^b r^-(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t)$
 - 8: $\mathbf{q}^t \leftarrow t^{-1} (\mathbf{q}^t - (\zeta_1^*(\alpha^t), \zeta_2^*(\beta^t)))$
 - 9: $\mathbf{r}^t \leftarrow t^{-1} \left((t-1) \cdot \mathbf{r}^{t-1} + \sum_{i=1}^b (r(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t)) \right)^2$
 - 10: $\alpha^t = \arg \min_{\alpha} \{\alpha \cdot \mathbf{r}^t - \zeta_1^*(\alpha)\}$ ▷ Inner Dual Step 1
 - 11: $\beta^t = \arg \min_{\beta} \{\beta \cdot \mathbf{r}^t - \zeta_2^*(\beta)\}$ ▷ Inner Dual Step 2
 - 12: $\gamma^t = \arg \min_{\gamma} \{\gamma \cdot \mathbf{q}^t - \Psi^*(\gamma)\}$ ▷ Outer Dual Step
 - 13: **end for**
 - 14: **return** \mathbf{w}^T
-

To address this challenge, we present **DENIM** (Algorithm 3) that itself nests its updates to mimic the nesting of the performance measures. **DENIM** follows a similar principle as **DUPLE** and is based on the **NEMESIS** algorithm of Kar et al. (2016). However, the **NEMESIS**

¹ $r(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t) = (r^+(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t), r^-(\mathbf{w}^t; \mathbf{x}_i^t, y_i^t))$.

Table 2 Statistics of datasets. Datasets are all binary and arranged in increasing order of the prevalence of the rare class. The most imbalanced dataset is KDD08 with under 1% positives and over 99% negatives. The most balanced is Cod-RNA with 33% positives and 64% negatives

Data set	# Points (K)	# Features	Rare class (%)	Source
KDDCup08	100	117	0.61	KDDCup08
PPI	240	85	1.19	Qi et al. (2006)
CoverType	580	54	1.63	UCI
Letter	20	16	3.92	UCI
IJCNN-1	140	22	9.57	UCI
Twitter	10	NA	22.6	SEMEVAL16
Adult (a9a)	50	123	23.93	UCI
Cod-RNA	490	8	33.30	UCI

algorithm faces the same drawbacks as the **SPADE** algorithm and is unsuitable for training deep models. Due to the more complex nature of the performance measure, **DENIM** works with a slightly different cost-weighted objective function at each time step.

$$h(\mathbf{w}; S, \alpha, \beta, \gamma) = (\gamma_1 \alpha_1 + \gamma_2 \beta_1) \cdot \hat{P}_S(\mathbf{w}) + (\gamma_1 \alpha_2 + \gamma_2 \beta_2) \cdot \hat{N}_S(\mathbf{w})$$

However, similar to **DUPLE**, all these coefficients can be calculated using cheap to update proxies. Note also that **DENIM** performs inner and outer dual updates that are themselves nested. **DENIM** enjoys similar convergence results as **DUPLE** which we omit for lack of space.

6 Experimental results

We performed extensive evaluation² of **DAME**, **DUPLE** and **DENIM** on benchmark and real-life challenge datasets and found it to outperform both traditional techniques for training neural networks, as well as the more nuanced training techniques popularly used to handle label imbalance and non-decomposable measures such as cost-weighted classification, plug-in methods and loss-augmented inference. We note that loss-augmented inference has been used in the past (Song et al. 2016) for ranking measures. We show that our methods outperform loss-augmented inference as well.

We developed two variants of the **DUPLE** and **DENIM** algorithms in a manner similar to Kar et al. (2016). **DUPLE-NS** refers to a variant of the **DUPLE** algorithm that uses count based rewards instead of sigmoidal rewards. Recall that **DUPLE** collects the sigmoidal rewards obtained by the algorithm on positive and negative points that are used to set the dual variables. **DUPLE-NS** instead checks whether those data points were classified correctly or not, i.e., it collects count rewards. However, while computing gradients, we revert back to the use of the sigmoidal reward function. **DENIMS-NS** was constructed similarly.

Datasets We use the datasets listed in Table 2. Twitter refers to the dataset revealed as a part of the SEMEVAL 2016 sentiment detection challenge (Esuli 2016).

Competing methods We implemented and adapted several benchmarks in an attempt to critically assess the performance of our methods.

² Code and datasets for our experiments are available at <https://github.com/purushottamkar/DeepPerf>.

- (1) **ANN 0-1** refers to a benchmark that trains a neural network using the cross-entropy loss function in an attempt to minimize the misclassification rate.
- (2) **ANN-p** refers to a benchmark that trains a neural network using a cost-weighted cross entropy loss function in an attempt to minimize cost-weighted misclassification loss. Each class is given a weight inversely proportional to its prevalence so as to give more emphasis to the rare class.
- (3) **STRUCT-ANN** refers to an implementation of loss-augmented inference and cutting plane techniques that have been popularly used in the past to optimize arbitrary non-decomposable performance measures. Please refer to [Appendix C](#) for the implementation details of this method.
- (4) **ANN-PG** refers to an implementation of the plug-in classification technique for F-measure as suggested in Koyejo et al. (2014).

Network architecture All neural networks being compared in a given experiment were given the same architecture. For experiments on concave and nested concave performance measures, all methods were provided with a network with 2 hidden layers of size 150 and 100 respectively. The ReLU activation function was used for all internal nodes and the output node used a sigmoidal activation. For experiments on pseudo-linear performance measures, the network contained three hidden layers of size 60, 60, 80 with sigmoidal activation on all nodes. The **DAME** method (Algorithm 1) used $d_{\text{int}} = 80$.

Batch size The batch size was fixed to 256 for all algorithms except **STRUCT-ANN** which was given a much larger batch size of 6000. This is because we found the method to do better with larger batch sizes. However, this greatly slowed down gradient computations and each iteration of **STRUCT-ANN** took much longer than each iteration of **DAME**. Increasing batch sizes indefinitely is not an option since it rapidly increases memory requirements as the number of temporary variables stored by popular implementations of back-propagation increase linearly with batch size. If one tries to reduce the memory footprint by making multiple forward passes, then the methods usually slow down excessively on GPU architectures. Even so we will see that our algorithms far outperform **STRUCT-ANN**. We note that even the past work of Song et al. (2016) did have to resort to small batches to avoid memory issues. **Step length** We used the Adam optimizer to tune the step lengths for all methods. The learning rate hyper-parameter provided to the Adam method was tuned in the range $\{0.001, 0.01, 0.1\}$ on a held-out validation set. Other hyper-parameters of the Adam method were fixed to their default values.

6.1 Experiments with pseudo-linear measures

Figure 1 below shows the performance of **DAME** on optimizing the F-measure. A naïve training method with the misclassification loss **ANN 0-1** yields extremely poor F-measure performance. Moreover, plug-in methods such as those proposed in Koyejo et al. (2014) **ANN-PG** also perform relatively poorly. **DAME** on the other hand is able to rapidly offer very good F-measure scores after looking at a fraction of the total data. **DAME** also outperforms **ANN 0-1** and **ANN-p** on all the datasets. We can also see that **STRUCT-ANN** offers consistently poor performance. This is because the method keeps predicting almost all points as negative, incurring a very low F-measure score. This may be partly due to using a smaller batch size. However, as pointed out before, **STRUCT-ANN** was provided a much larger batch size than other methods. Increasing the batch size much further adversely affected performance due to excessive memory usage.

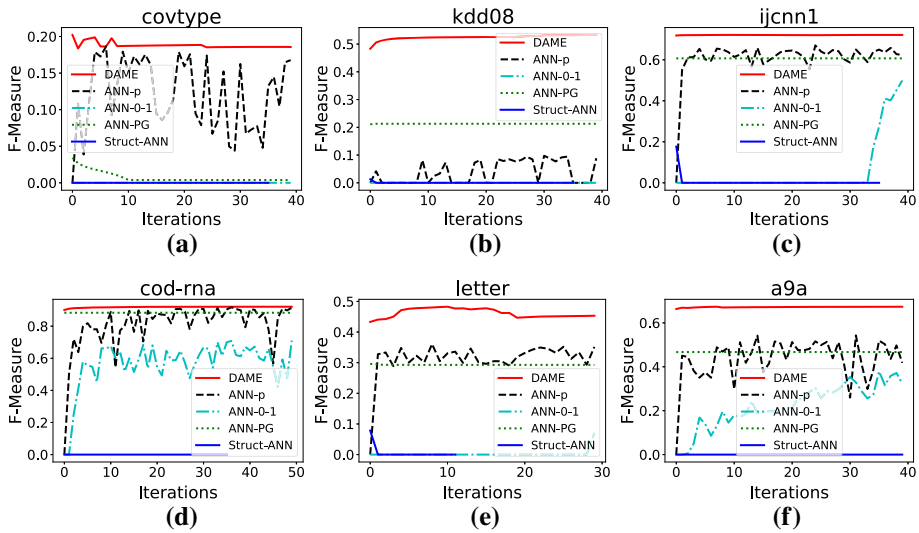


Fig. 1 Experiments on maximizing F-measure, a pseudo-linear performance measure. The Y-axis represents test error and the X-axis represents training iterations. We paused the training after every few iterations and recorded the test accuracy of the model at that snapshot. **DAME** outperforms all other benchmarks by a large margin. **STRUCT-ANN** tends to classify most data points as negative and gets a poor F-measure score as a result

6.2 Experiments on concave measures

Figure 2a–d on optimizing MinTPRTNR and Fig. 2e–h on optimizing QMean show that **DUPLE** offers faster convergence in comparison to **ANN 0-1** which has a very hard time obtaining a non-trivial MinTPRTNR score. For the experiment on IJCNN1, we ran the experiment for a longer time to allow **ANN 0-1** and **STRUCT-ANN** to converge and we observe that they are highly time intensive, when compared to **DUPLE**.

In the experiments with MinTPRTNR, both **DUPLE** and **ANN-p** perform comparably though **ANN-p** gradually starts overfitting whereas **DUPLE** retains its performance. With QMean, **ANN-p** starts overfitting on both PPI and KDD08 though **DUPLE** keeps performing better. **DUPLE-NS** is slightly slower than **ANN-p** on IJCNN1. Our experiments show that **DUPLE** and **DUPLE-NS** are more consistent and robust to overfitting across datasets than **ANN-p**.

DUPLE and its variant **DUPLE-NS** outperform most competitors both in terms of speed as well as accuracy though both **DUPLE** and **DUPLE-NS** perform comparably. It is also to be noted that **DUPLE** not only takes fewer iterations than **STRUCT-ANN**, but each iteration of **DUPLE** much faster than that of **STRUCT-ANN** since we gave **STRUCT-ANN** a batch size of 6000 whereas **DUPLE** operated with batch sizes of 256, which is an order of magnitude smaller. Thus, **STRUCT-ANN** is even slower in convergence than these figures indicate.

6.3 Experiments with nested performance measures

In Fig. 2i–l, we can see the results obtained by **DENIM** while optimizing the KLD performance measure. It shows rapid convergence to near-perfect quantification scores. The experiments also show that **DENIM** and **DENIMS-NS** require far fewer iterations than its

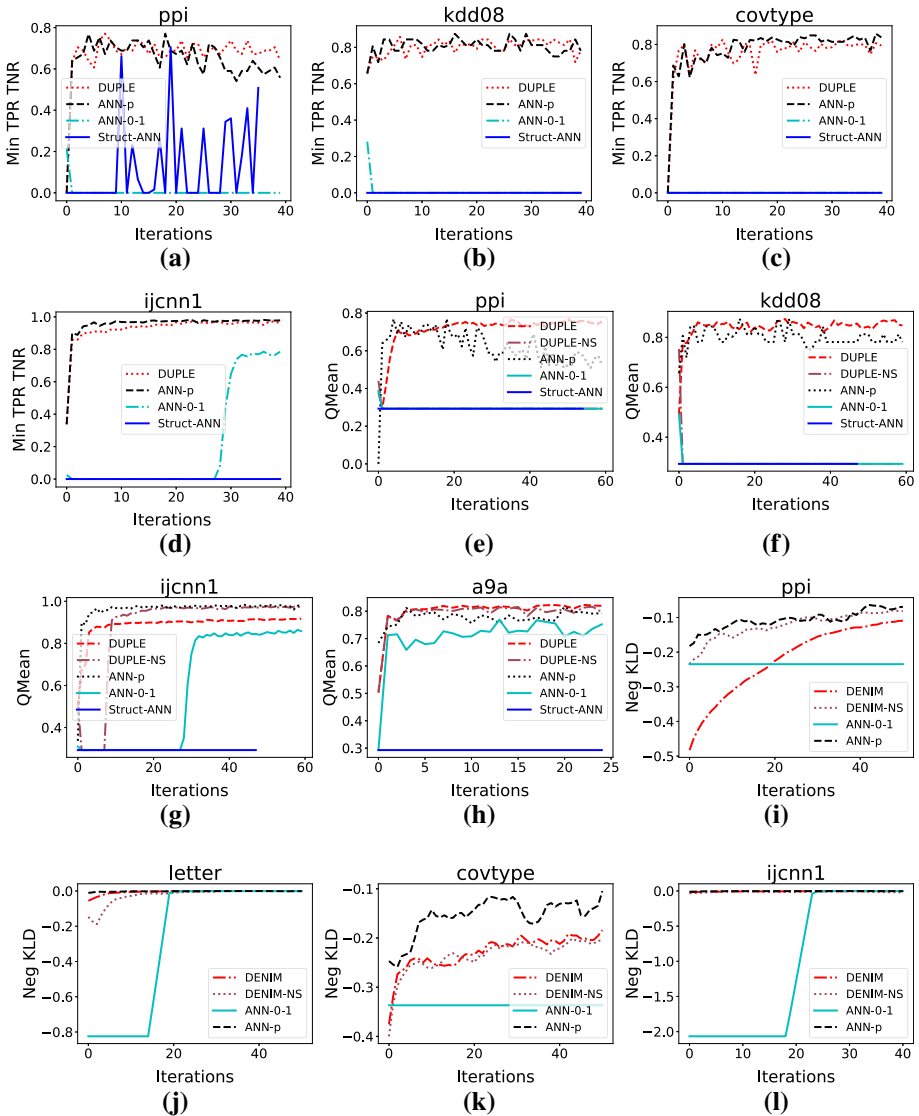


Fig. 2 Experiments with MinTPRTNR (Fig. 2a–d) and QMean (Fig. 2e–h), two concave performance measures and the KL divergence (Fig. 2i–l), a nested concave performance measure. The Y-axis denotes test error and the X-axis denotes training iterations. We paused training after every few iterations and recorded the test accuracies of the models presented by various methods at that time. **DUPLE** and **ANN-p** are the leading methods for MinTPRTNR although **DUPLE** offers more stable performance e.g. on PPI. For the QMean performance measure, **DUPLE** leads on PPI, KDD08 and A9A and offers more stable performance than **ANN-p**. **DUPLE** and **ANN-p** are the leading methods for the KL divergence with **ANN-p** leading on the CovType dataset. On most experiments, **STRUCT-ANN** continues to classify most data points as negative and gets a poor score with respect to all performance measures as a result. On the PPI dataset with MinTPRTNR, **STRUCT-ANN** does offer non-trivial predictions, however, its behavior is very erratic

competitor **ANN 0-1** (whenever **ANN 0-1** is successful at all). The **STRUCT-ANN** benchmark does not seem to appear in the graphs for this performance measure since it always

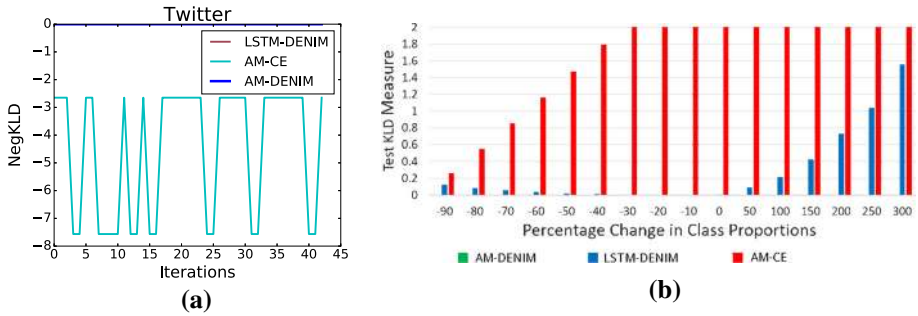


Fig. 3 Results on the Twitter sentiment analysis task. **a** Convergence to optimal test KLD performance for different RNN models. **b** Change in quantification performance with distribution drift

got a very large KL-divergence value by consistently predicting every data point as negative. **ANN-p** achieves comparable performance to **DENIM** on most datasets and is the winner on the CoverType dataset. However, we believe that with proper pretraining, we can get much better performance for **DENIM** on this performance measure.

6.4 Case study: quantification for sentiment analysis

We report the results of experiments comparing the performance of the **DENIM** on a Twitter sentiment detection challenge problem. The task in this challenge was to correctly ascertain the fraction of tweets exhibiting various sentiments. The performance was measured using the Kullback–Leibler divergence (1). We trained an end-to-end LSTM model trained using **DENIM**. We also trained an attention-enabled network for the same task using **DENIM**. Our models accepted raw text in the standard one-hot encoding format and performed task specific optimization and generated task specific vocabulary embeddings. Our representations were 64-dimensional and learnt jointly with other network parameters.

Implementation details All our LSTM models used a single hidden layer with 64 hidden nodes, which gave rise to 64-dimensional hidden state representations. For the LSTM model, the final label was obtained by applying a linear model with a sigmoidal activation function. For the attention models (referred to as **AM**), the decoder hidden states were set to be 64-dimensional as well. The alignment model was set to be a feed-forward model with a softmax layer. Step lengths were tuned using standard implementations of the Adam method. Training was done by adapting the **DENIM** method.

DENIM is able to obtain near perfect quantification on both LSTM ($KLD = 0.007$) as well as AM ($KLD = 0.00002$) models (see Fig. 3a). In contrast, the classical cross-entropy method with attention model (**AM-CE**) is unable to obtain satisfactory performance. **DENIM** converges to optimal test KLD performance in not only far lesser iterations, but also by using far fewer data samples. Also note that the **AM** models trained with **DENIM** give KLD losses that are much smaller than what they offer when trained with **DENIM**.

We also experimented with artificially changing the fraction of positive and negative examples in order to see the performance of our model under distribution drift (see Fig. 3b). The fraction of negatives and positives in the test set was distorted from their original values by re-sampling. As the test distribution priors are distorted more and more, **AM-CE** (Attention Model trained with Cross Entropy) performs extremely poorly. **DENIM** with LSTMs displays some degree of robustness to drift but succumbs at extremely high level of drift. **DENIM** with

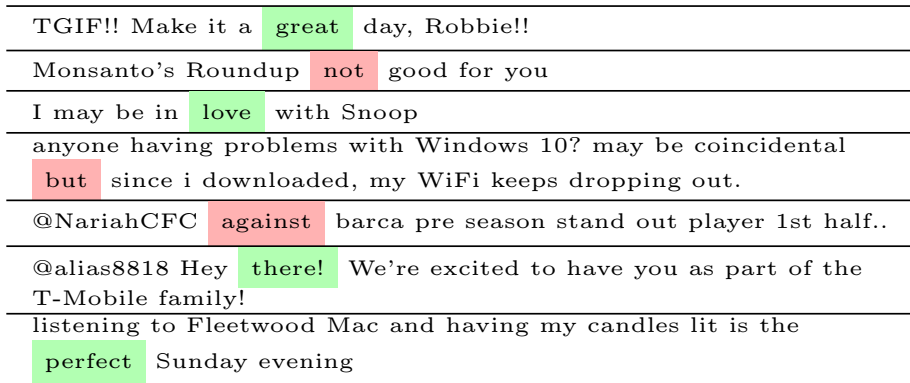


Fig. 4 Figuring where the attention is. Highlighted words got high attention scores. A red (green) highlight indicates that the tweet was tagged with a negative (positive) sentiment (Color figure online)

AM models on the other hand, remains extremely robust to even a high degree of distribution drift, offering near-zero KLD error.

The benefits of the attention models employed by **DENIM** allow it to identify critical words in a tweet that signal its polarity. The highlighted words (see Fig. 4) are those for which **DENIM** assigned an attention score $\alpha \approx 1$.

7 Conclusion

Our work presents algorithms to train neural networks and other non-linear models in pursuit of non-decomposable performance measures that are popularly used in label-imbalanced training and quantification tasks. Our algorithms offer better performance while using fewer iterations/samples, as compared to traditional cross-entropy based training as well as several other benchmarks usually adopted while optimizing non-decomposable losses such as cost-weighted classification or plug-in methods or loss-augmented inference. This leads to several avenues of future work and improvements. We observed impressive performance boosts when we employed pretraining with the **DAME** method and the same can be experimented with other methods as well. It would be very interesting to investigate extensions of the **DUPLE**, **DENIM** and **DAME** methods to more complex prediction tasks such as multi-class and multi-label classification tasks. Obtaining a better theoretical understanding of how these methods behave when operated with non-linear models (in particular neural networks) would also be of interest.

Acknowledgements A.S. did this work while he was a student at IIT Kanpur and acknowledges support from The Alan Turing Institute under the Turing Doctoral Studentship grant TU/C/000023. P. Kar is supported by the Deep Singh and Daljeet Kaur Faculty Fellowship and the Research-I foundation at IIT Kanpur, and thanks Microsoft Research India and Tower Research for research grants.

Appendix A: Proof of Theorem 1

To prove Theorem 1, we first show the following result. As mentioned before, we will omit the lower layers from the analysis since they are not affected by the fine-tuning phase where **DAME** is executed.

Lemma 1 *If executed with a uniform step length satisfying $\eta < \frac{2}{L\kappa}$ with a batch size of B , then within $\frac{\kappa^2}{m} \frac{1}{\eta(1-\frac{L\kappa\eta}{2})\epsilon^2}$ iterations, **DAME** identifies a model $\mathbf{w}_1^{t,0}$ such that $\|\nabla_{\mathbf{w}} V_{\tilde{S}^t}(\mathbf{w}_1^{t,0}, v^{t-1})\|_2 \leq \epsilon + \mathcal{O}\left(\frac{r}{\sqrt{B}}\right)$ with high probability. If a mini-batch is not used and we have $S_{t,i} = \tilde{S}$ for all time steps t, i , then we are (deterministically) assured of model such that $\|\nabla_{\mathbf{w}} V_{\tilde{S}^t}(\mathbf{w}_1^{t,0}, v^{t-1})\|_2 \leq \epsilon$.*

Given this, Theorem 1 follows as shown below.

Proof (of Theorem 1) Since we have $\nabla_{\mathbf{w}} \mathcal{P}_{(\mathbf{a},\mathbf{b})}(\mathbf{w}) = \frac{\nabla_{\mathbf{w}} V(\mathbf{w}, \mathcal{P}_{(\mathbf{a},\mathbf{b})})}{\mathcal{P}_{\mathbf{b}}(\mathbf{w})}$, and $\mathcal{P}_{\mathbf{b}}(\mathbf{w}) \geq m$, Theorem 1 we have $\|\nabla_{\mathbf{w}} \mathcal{P}_{(\mathbf{a},\mathbf{b})}(\mathbf{w})\|_2 \leq \frac{1}{m} \|\nabla_{\mathbf{w}} V(\mathbf{w}, \mathcal{P}_{(\mathbf{a},\mathbf{b})})\|_2$. Since Lemma 1 gives us $\|\nabla_{\mathbf{w}} V(\mathbf{w}, \mathcal{P}_{(\mathbf{a},\mathbf{b})})\|_2 \leq \epsilon + \mathcal{O}\left(\frac{r}{\sqrt{B}}\right)$, we have $\|\nabla_{\mathbf{w}} \mathcal{P}_{(\mathbf{a},\mathbf{b})}(\mathbf{w})\|_2 \leq \frac{\epsilon}{m} + \mathcal{O}\left(\frac{r}{m\sqrt{B}}\right)$. Setting $\epsilon' = \frac{\epsilon}{m}$ gives us the desired result. \square

Proof (of Lemma 1) It is easy to see that $V(\mathbf{w}_1, v)$ is a $L\kappa$ -strongly smooth function of the model parameter \mathbf{w}_1 for any realizable valuation i.e. $v = \mathcal{P}_{(\mathbf{a},\mathbf{b})}(\mathbf{w})$ for some $\mathbf{w} \in \mathcal{W}$. Now, the **DAME** algorithm makes the following model updates within the inner loop

$$\mathbf{w}_1^{t-1,t'} = \mathbf{w}_1^{t-1,t'-1} + \eta \cdot \mathbf{g}^{(t-1,t')},$$

where $\mathbf{g}^{(t-1,t')} = \nabla_{\mathbf{w}_1^{t-1,t'-1}} V(\mathbf{w}_1^{t-1,t'-1}, v^t)$ if not using a mini batch and $\mathbf{g}^{(t-1,t')}$ is the gradient calculated on the mini-batch if using one. Since the negated valuation function i.e. $-V$ is strongly smooth, we get

$$\begin{aligned} V(\mathbf{w}_1^{t-1,t'}, v^t) &\geq V(\mathbf{w}_1^{t-1,t'-1}, v^t) + \left\langle \mathbf{g}^{(t-1,t')}, \mathbf{w}_1^{t-1,t'} - \mathbf{w}_1^{t-1,t'-1} \right\rangle \\ &\quad - \frac{L\kappa}{2} \left\| \mathbf{w}_1^{t-1,t'} - \mathbf{w}_1^{t-1,t'-1} \right\|_2^2 \\ &= V(\mathbf{w}_1^{t-1,t'-1}, v^t) + \eta \left(1 - \frac{L\kappa\eta}{2} \right) \left\| \mathbf{g}^{(t-1,t')} \right\|_2^2 \end{aligned}$$

Now this shows that at each step where $\|\mathbf{g}^{(t-1,t')}\| > \epsilon$, the valuation of the model $\theta^{(t+1,i)}$ goes up by at least $\eta \left(1 - \frac{L\kappa\eta}{2} \right) \epsilon^2$. It is easy to see that if $V(\mathbf{w}_1^t, v^t) \geq c$ then $\mathcal{P}(\mathbf{w}_1^t) \geq \mathcal{P}(\mathbf{w}_1^{t-1}) + \frac{c}{M}$. Since the maximum value of the performance measure for any model is $\frac{M}{m}$, putting these results together tell us that **DAME** cannot execute more than $\frac{M^2}{m} \frac{1}{\eta(1-\frac{L\kappa\eta}{2})\epsilon^2}$

inner iterations without encountering a model $\mathbf{w}_1^{t,t'}$ such that $\|\mathbf{g}^{(t-1,t')}\|_2 \leq \epsilon$.

If not using a mini-batch, this already proves the theorem. Otherwise we have to do a bit more work in applying a standard McDiarmid’s inequality to control the norm deviation. Let $S := S_{t,i} = \{(\mathbf{x}_{j_i}, y_{j_i})\}_{j=1,\dots,B}$ denote the mini-batch chosen and let \mathbf{g}_S denote the gradient with respect to the valuation function obtained as a result. Also denote $\nabla = \nabla_{\mathbf{w}_1^{t-1,t'-1}} V(\mathbf{w}_1^{t-1,t'-1}, v^t)$ for notational simplicity..

Notice that the mini-batches are chosen uniformly randomly (with replacement) and independently of previous choices of mini-batches. This assures us that $\mathbb{E}[\mathbf{g}_S] = \nabla$ where the expectation is over the choice of the mini-batch S . Let $Z_S := \|\mathbf{g}_S - \nabla\|_2$. Suppose we had instead chosen a mini-batch which differs from S at just one data point i.e. $S' = S \setminus \{(\mathbf{x}_{j_k}, y_{j_k})\} \cup \{(\mathbf{x}'_{j_k}, y'_{j_k})\}$ for some $k \in [B]$.

Then, by triangle inequality, we have $Z_S - Z_{S'} \leq \|\mathbf{g}_S - \mathbf{g}_{S'}\| \leq \frac{r}{B}$ since we have assumed bounded gradients with respect to the reward functions. Similarly, we have $Z_{S'} - Z_S \leq \frac{r}{B}$ which gives us $|Z_S - Z_{S'}| \leq \frac{r}{B}$ which tells us that Z_S is a $\frac{r}{B}$ -stable estimator. Moreover, similarly and by applying Jensen’s inequality and exploiting the fact that the mini-batch is chosen i.i.d., we see that $\mathbb{E}[Z_S] \leq \sqrt{\mathbb{E}[Z_S^2]} \leq \frac{2r}{\sqrt{B}}$. An application of the McDiarmid’s inequality gives us, with probability at least $1 - \delta$,

$$\|\mathbf{g}^{(t-1,t')} - \nabla^{(t-1,t')}\|_2 \leq \frac{2r}{\sqrt{B}} + r\sqrt{\frac{\log \frac{1}{\delta}}{B}}$$

Taking a union bound over all iterations and applying the triangle inequality finishes the proof as $\|\nabla^{(t-1,t')}\|_2 \leq \|\mathbf{g}^{(t-1,t')}\|_2 + \|\mathbf{g}^{(t-1,t')} - \nabla^{(t-1,t')}\|_2 \leq \epsilon + \mathcal{O}\left(r\sqrt{\frac{\log \frac{1}{\delta}}{B}}\right)$. \square

Appendix B: Proof of Theorem 2

Theorem 3 Consider a concave performance measure defined using a link function Ψ that is concave and whose negation is L' -strongly smooth. Then, if executed with a uniform step length satisfying $\eta < \frac{2}{L'}$, then **DUPLE** ϵ -stabilizes within $\tilde{\mathcal{O}}\left(\frac{1}{\epsilon^2}\right)$ iterations. More specifically, within T iterations, **DUPLE** identifies a model \mathbf{w}^t such that $\|\nabla^t\|_2 \leq \mathcal{O}\left(\sqrt{L' \frac{\log T}{T}}\right)$.

Proof Recall that we also assume that the negated reward functions $-r(f(\mathbf{x}; \mathbf{w}), y)$ are L' -strongly smooth functions of the model \mathbf{w} (which the negated sigmoid does satisfy for $L' = \mathcal{O}(1)$). However we will not assume convexity or concavity of the loss function. We will use the shorthands $\nabla^t = \nabla_{\mathbf{w}} g(\mathbf{w}^t; S_t, \alpha^t, \beta^t)$ and $F(\mathbf{w}^t, \alpha^t) = g(\mathbf{w}^t; S_t, \alpha^t, \beta^t)$ where $\alpha^t = (\alpha^t, \beta^t)$. Also recall that we have $g(\mathbf{w}; S_t, \alpha, \beta) = \alpha \cdot \hat{P}_{S_t}(\mathbf{w}) + \beta \cdot \hat{N}_{S_t}(\mathbf{w})$ as the cost-weighted objective function with respect to which gradients are taken. We note that we are giving the proof for the case when mini-batches are taken and not assuming access to full-batches.

DUPLE makes the following model update $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \cdot \nabla^t$. Using the strong smoothness of the reward functions, we get

$$F(\mathbf{w}^{t+1}, \alpha^t) \geq F(\mathbf{w}^t, \alpha^t) + \langle \nabla^t, \mathbf{w}^{t+1} - \mathbf{w}^t \rangle - \frac{L'}{2} \|\mathbf{w}^{t+1} - \mathbf{w}^t\|_2^2,$$

which, upon rearranging, give us $\|\nabla^t\|_2^2 \leq \frac{F(\mathbf{w}^{t+1}, \alpha^t) - F(\mathbf{w}^t, \alpha^t)}{\eta(1 - \frac{\eta L'}{2})}$, which, upon summing up, gives us

$$\sum_{t=1}^T \|\nabla^t\|_2^2 \leq \frac{1}{\eta\left(1 - \frac{\eta L'}{2}\right)} \left(F(\mathbf{w}^{T+1}, \alpha^T) + \sum_{t=2}^T F(\mathbf{w}^t, \alpha^{t-1}) - F(\mathbf{w}^t, \alpha^t) \right).$$

Now, we notice that since the (negated) reward functions are L' -strongly smooth, so is the entire (negated) performance measure $\mathcal{P}_{\Psi, S}(\mathbf{w})$. However, this implies (using the self duality of the L_2 norm) that (see for example Kakade et al. 2012 Theorem 3) that the function $\Psi^*(\alpha, \beta)$ is $\frac{1}{L'}$ -strongly concave (in other words $-\Psi^*$ is strongly convex) i.e. it satisfies (denote $\alpha = (\alpha, \beta)$)

$$\Psi^*(\alpha) \leq \Psi^*(\tilde{\alpha}) + \langle \nabla_{\alpha} \Psi^*(\tilde{\alpha}), \alpha - \tilde{\alpha} \rangle - \frac{1}{2L'} \|\alpha - \tilde{\alpha}\|_2^2$$

Thus, the functions $F(\mathbf{w}, \alpha)$ are $\frac{1}{L'}$ strongly concave with respect to α . Now, notice that **DUPL**E performs its dual step using an arg min operation on the accumulated rewards on positive and negative points, with $-\Psi^*$ acting as a regularizer. This is exactly a follow the regularized leader step and we have shown above that $-\Psi^*$ is strongly convex since Ψ^* is strongly concave. Thus, by a standard forward regret-analysis (see for example (Kar et al. 2014) Theorem 1) of the dual updates on the strongly concave functions $F(\mathbf{w}, \alpha)$, we get

$$\sum_{t=2}^T F(\mathbf{w}^t, \alpha^{t-1}) - F(\mathbf{w}^t, \alpha^t) \leq \mathcal{O}(L' \log T).$$

Thus, we get $\sum_{t=1}^T \|\nabla^t\|_2^2 \leq \mathcal{O}\left(\frac{L' \log T}{\eta(1-\frac{\eta L'}{2})}\right)$. This means within T iterations, we must come across a point where we have $\|\nabla^t\|_2^2 \leq \frac{1}{T} \mathcal{O}\left(\frac{L' \log T}{\eta(1-\frac{\eta L'}{2})}\right)$ which proves the result. \square

Appendix C: Implementation details for the **STRUCT-ANN** benchmark

Our goal here is to give a brief overview of the techniques used to perform loss-augmented inference to build the **STRUCT-ANN** benchmark. We refer the reader to Tsochantaridis et al. (2005) and Joachims et al. (2009) for the motivation behind these techniques and more details. For this discussion, we will work with a non-decomposable loss function Δ which is a function of the TPR, TNR values of the classifier i.e. for a classifier $h : \mathcal{X} \rightarrow \{-1, +1\}$ we have $\Delta(h) = \Delta(\text{TPR}(h), \text{TNR}(h))$. In order to apply this to performance measures like F-measure which are ‘‘reward’’ like and which we wish to maximize instead, simply take $\Delta = -\text{F-measure}$ i.e. negate the performance measure to get something that looks like a loss function.

Suppose we are trying to train a scoring function $f_{\mathbf{w}} : \mathcal{X} \rightarrow \mathbb{R}$ indexed by a parameter \mathbf{w} using a training set or mini-batch of $(\mathbf{x}_i, y_i)_{i=1, \dots, n}$ of n training points. Let $\mathbf{y} = \{y_1, y_2, \dots, y_n\} \in \{-1, +1\}^n$ denote the training label vector. Then, for any other label vector $\hat{\mathbf{y}} = \{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n\} \in \{-1, +1\}^n$ of the same dimension, we define

$$\begin{aligned} \text{TPR}(\hat{\mathbf{y}}, \mathbf{y}) &:= \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{\hat{y}_i = 1 \wedge y_i = 1\} \\ \text{TNR}(\hat{\mathbf{y}}, \mathbf{y}) &:= \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{\hat{y}_i = -1 \wedge y_i = -1\}, \end{aligned}$$

where $\mathbb{I}\{\cdot\}$ denotes the indicator function which outputs 1 if the argument is true else 0. Then, the structural surrogate for the performance measure Δ , which we denote by the notation $\hat{\Delta}$ is defined for any scoring function such as $f_{\mathbf{w}}$ as follows

$$\hat{\Delta}(f_{\mathbf{w}}) = \max_{\hat{\mathbf{y}} \in \{-1, +1\}^n} \left\{ \Delta(\text{TPR}(\hat{\mathbf{y}}, \mathbf{y}), \text{TNR}(\hat{\mathbf{y}}, \mathbf{y})) + \sum_{i=1}^n (\hat{y}_i - y_i) \cdot f_{\mathbf{w}}(\mathbf{x}_i) \right\}$$

By applying Danskin’s theorem, we can conclude that for a given parameter \mathbf{w} , if $\tilde{\mathbf{y}}$ is a maximizer of the above expression i.e. if

$$\tilde{\mathbf{y}} = \arg \max_{\hat{\mathbf{y}} \in \{-1,+1\}^n} \left\{ \Delta(\text{TPR}(\hat{\mathbf{y}}, \mathbf{y}), \text{TNR}(\hat{\mathbf{y}}, \mathbf{y})) + \sum_{i=1}^n (\hat{y}_i - y_i) \cdot f_{\mathbf{w}}(\mathbf{x}_i) \right\},$$

then a subgradient to the function $\hat{\Delta}(\cdot)$ can be found at a parameter \mathbf{w}^0 as follows

$$\frac{\partial \hat{\Delta}}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}^0} \ni \sum_{i=1}^n (\tilde{y}_i - y_i) \cdot \frac{\partial f_{\mathbf{w}}(\mathbf{x}_i)}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}^0} = \frac{\partial}{\partial \mathbf{w}} \left(\sum_{i=1}^n (\tilde{y}_i - y_i) \cdot f_{\mathbf{w}}(\mathbf{x}_i) \right) \Big|_{\mathbf{w}=\mathbf{w}^0}$$

which tells us that we can perform (sub)gradient descent with respect to the loss function $\hat{\Delta}(\cdot)$ using a “weighted” back-propagation step with weights equal to $(\tilde{y}_i - y_i)$ once we have identified $\tilde{\mathbf{y}}$.

Thus, in order to perform a single (sub)gradient descent step, we have to solve the following problem

$$\arg \max_{\hat{\mathbf{y}} \in \{-1,+1\}^n} \left\{ \Delta(\text{TPR}(\hat{\mathbf{y}}, \mathbf{y}), \text{TNR}(\hat{\mathbf{y}}, \mathbf{y})) + \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i) \cdot f_{\mathbf{w}}(\mathbf{x}_i) \right\}$$

To do this, first we need the values $f_{\mathbf{w}}(\mathbf{x}_i)$ for all data points in the training set or mini-batch which can be done using a single forward pass. From hereon it is easy to see that the above optimization problem is equivalent to the following expressions

$$\begin{aligned} & \arg \max_{a,b} \left\{ \arg \max_{\substack{\hat{\mathbf{y}}:\text{TPR}(\hat{\mathbf{y}},\mathbf{y})=a, \\ \text{TNR}(\hat{\mathbf{y}},\mathbf{y})=b}} \left\{ \Delta(\text{TPR}(\hat{\mathbf{y}}, \mathbf{y}), \text{TNR}(\hat{\mathbf{y}}, \mathbf{y})) + \frac{1}{n} \sum_{i=1}^n \hat{y}_i \cdot f_{\mathbf{w}}(\mathbf{x}_i) \right\} \right\} \\ & \arg \max_{a,b} \left\{ \arg \max_{\substack{\hat{\mathbf{y}}:\text{TPR}(\hat{\mathbf{y}},\mathbf{y})=a, \\ \text{TNR}(\hat{\mathbf{y}},\mathbf{y})=b}} \left\{ \Delta(a, b) + \frac{1}{n} \sum_{i=1}^n \hat{y}_i \cdot f_{\mathbf{w}}(\mathbf{x}_i) \right\} \right\} \\ & \arg \max_{a,b} \left\{ \Delta(a, b) + \arg \max_{\substack{\hat{\mathbf{y}}:\text{TPR}(\hat{\mathbf{y}},\mathbf{y})=a, \\ \text{TNR}(\hat{\mathbf{y}},\mathbf{y})=b}} \left\{ \frac{1}{n} \sum_{i=1}^n \hat{y}_i \cdot f_{\mathbf{w}}(\mathbf{x}_i) \right\} \right\} \end{aligned}$$

Note that the outer maximum over a, b need only be explored over discrete values of a and b since the only valid values of a, b are $\{0, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}, 1\}$ since they are supposed to be valid TPR, TNR values. Now, a naive way to solve the above would take $\mathcal{O}(n^2)$ time where n is the training set size or mini-batch size, by cycling over all possible values of a, b . However, for several performance measures Δ which includes all measures we have studied, this optimization can be carried out in $\mathcal{O}(n \log n)$ time. We refer the reader to Kar et al. (2014) for details.



References

Barranquero, J., Díez, J., & del Coz, J. J. (2015). Quantification-oriented learning based on reliable classifiers. *Pattern Recognition*, 48(2), 591–604.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., & Bengio, Y. (2010). Theano: A CPU and GPU math compiler in Python. In *Proceedings of the 9th Python in science conference (SciPy 2010)* (pp. 1–7). Austin, USA.

- Eban, E., Schain, M., Mackey, A., Gordon, A., Saurous, R., & Elidan, G. (2017). Scalable Learning of non-decomposable objectives. In *Proceedings of the 20th international conference on artificial intelligence and statistics (AISTATS)*.
- Esuli, A. (2016). ISTI-CNR at SemEval-2016 Task 4: Quantification on an ordinal scale. In *Proceedings of the 10th international workshop on semantic evaluation (SemEval 2016)*. San Diego, US.
- Esuli, A., & Sebastiani, F. (2015). Optimizing text quantifiers for multivariate loss functions. *ACM Transactions on Knowledge Discovery and Data* 9(4), Article 27. <https://doi.org/10.1145/2700406>.
- Gao, W., & Sebastiani, F. (2015). Tweet sentiment: From classification to quantification. In *Proceedings of the 7th international conference on advances in social network analysis and mining (ASONAM 2015)* (pp. 97–104). Paris, FR.
- Joachims, T., Finley, T., & Yu, C. N. J. (2009). Cutting-plane training of structural SVMs. *Machine Learning Journal*, 77(1), 27–59.
- Kakade, S., Shalev-Shwartz, S., & Tewari, A. (2012). Regularization techniques for learning with matrices. *Journal of Machine Learning Research*, 13, 1865–1890.
- Kar, P., Li, S., Narasimhan, H., Chawla, S., & Sebastiani, F. (2016). Online optimization methods for the quantification problem. In *Proceedings of the 22nd ACM international conference on knowledge discovery and data mining (SIGKDD 2016)* (pp. 1625–1634). San Francisco, USA.
- Kar, P., Narasimhan, H., & Jain, P. (2014). Online and stochastic gradient methods for non-decomposable loss functions. In *Proceedings of the 28th annual conference on neural information processing systems (NIPS 2014)* (pp. 694–702). Montreal, USA.
- Kar, P., Narasimhan, H., & Jain, P. (2015). Surrogate functions for maximizing precision at the top. In *Proceedings of the 32nd international conference on machine learning (ICML 2015)* (pp. 189–198). Lille, FR.
- Kar, P., Sriperumbudur, B.K., Jain, P., & Karmick, H. (2013). On the generalization ability of online learning algorithms for pairwise loss functions. In *30th international conference on machine learning (ICML)*.
- Kennedy, K., Namee, B.M., & Delany, S.J. (2010). Learning without default: A study of one-class classification and the low-default portfolio problem. In *International conference on artificial intelligence and cognitive science (ICAICS)*, *Lecture notes in computer science* (Vol. 6202, pp. 174–187).
- Koyejo, O.O., Natarajan, N., Ravikumar, P.K., & Dhillon, I.S. (2014). Consistent binary classification with generalized performance metrics. In *Proceedings of the 28th annual conference on neural information processing systems (NIPS 2014)* (pp. 2744–2752). Montreal, CA.
- Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge: Cambridge University Press.
- Narasimhan, H., & Agarwal, S. (2013). A structural SVM based approach for optimizing partial AUC. In *30th international conference on machine learning (ICML)*.
- Narasimhan, H., & Agarwal, S. (2013). SVM_{pAUC}^{tight}: A new support vector method for optimizing partial AUC based on a tight convex upper bound. In *ACM SIGKDD conference on knowledge, discovery and data mining (KDD)*.
- Narasimhan, H., Kar, P., & Jain, P. (2015). Optimizing non-decomposable performance measures: A tale of two classes. In *Proceedings of the 32nd international conference on machine learning (ICML 2015)* (pp. 199–208). Lille, FR.
- Narasimhan, H., Vaish, R., & Agarwal, S. (2014). On the statistical consistency of plug-in classifiers for non-decomposable performance measures. In *28th annual conference on neural information processing systems (NIPS)*.
- Qi, Y., Bar-Joseph, Z., & Klein-Seetharaman, J. (2006). Evaluation of different biological data and computational classification methods for use in protein interaction prediction. *Proteins*, 63, 490–500.
- Schäfer, D., & Hüllermeier, E. (2018). Dyad ranking using Plackett–Luce models based on joint feature representations. *Machine Learning*, 107(5), 903–941.
- Song, Y., Schwing, A.G., Zemel, R.S., & Urtasun, R. (2016). Training deep neural networks via direct loss minimization. In *Proceedings of the 33rd international conference on machine learning (ICML)*.
- Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large Margin methods for structured and interdependent output variables. *Journal of Machine Learning*, 6, 1453–1484.
- Vincent, P. (1994). *An introduction to signal detection and estimation*. New York: Springer.

Affiliations

Amartya Sanyal^{1,2}  · **Pawan Kumar**³ · **Purushottam Kar**³  · **Sanjay Chawla**⁴ · **Fabrizio Sebastiani**⁵

Sanjay Chawla
schawla@qf.org.qa

Fabrizio Sebastiani
fabrizio.sebastiani@isti.cnr.it

- ¹ The University of Oxford, Oxford, UK
- ² The Alan Turing Institute, London, UK
- ³ Indian Institute of Technology Kanpur, Kanpur, India
- ⁴ Qatar Computing Research Institute, HBKU, Doha, Qatar
- ⁵ Istituto di Scienza e Tecnologia dell'Informazione, Pisa, Italy