KDGAN: Knowledge Distillation with Generative Adversarial Networks

Xiaojie Wang
University of Melbourne
xiaojiew94@gmail.com

Rui Zhang*
University of Melbourne
rui.zhang@unimelb.edu.au

Yu Sun
Twitter Inc.
ysun@twitter.com

Jianzhong Qi
University of Melbourne
jianzhong.qi@unimelb.edu.au

Abstract

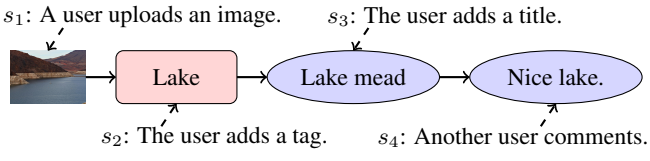
Knowledge distillation (KD) aims to train a lightweight classifier suitable to provide accurate inference with constrained resources in multi-label learning. Instead of directly consuming feature-label pairs, the classifier is trained by a teacher, i.e., a high-capacity model whose training may be resource-hungry. The accuracy of the classifier trained this way is usually suboptimal because it is difficult to learn the true data distribution from the teacher. An alternative method is to adversarially train the classifier against a discriminator in a two-player game akin to generative adversarial networks (GAN), which can ensure the classifier to learn the true data distribution at the equilibrium of this game. However, it may take excessively long time for such a two-player game to reach equilibrium due to high-variance gradient updates. To address these limitations, we propose a three-player game named KDGAN consisting of a classifier, a teacher, and a discriminator. The classifier and the teacher learn from each other via distillation losses and are adversarially trained against the discriminator via adversarial losses. By simultaneously optimizing the distillation and adversarial losses, the classifier will learn the true data distribution at the equilibrium. We approximate the discrete distribution learned by the classifier (or the teacher) with a concrete distribution. From the concrete distribution, we generate continuous samples to obtain low-variance gradient updates, which speed up the training. Extensive experiments using real datasets confirm the superiority of KDGAN in both accuracy and training speed.

1 Introduction

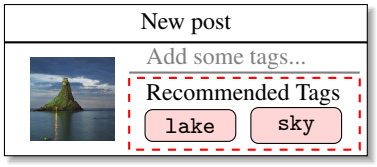
In machine learning, it is common that more resources such as input features [47] or computational resources [23], which we refer to as *privileged provision*, are available at the stage of training a model than those available at the stage of running the deployed model (i.e., the inference stage). Figure 1 shows an example application of image tag recommendation, where more input features (called *privileged information* [47]) are available at the training stage than those available at the inference stage. Specifically, the training stage has access to images as well as image titles and comments (textual information) as shown in Figure 1a, whereas the inference stage only has access to images themselves as shown in Figure 1b. After a smart phone user uploads an image and is about to provide tags for the image, it is inconvenient to type tags on the phone and thinking about tags for the image also takes time, so it is very useful to recommend tags based on the image as shown in Figure 1b. Another example application is unlocking mobile phones by face recognition. We usually deploy face

*Corresponding author

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.



(a) Training: After a user uploads an image, additional text such as comments and titles besides the labeled tags is accumulated.



(b) Inference: We recommend `bay` and `sky` right after an image is uploaded.

Figure 1: Image tag recommendation where the additional text is only available for training.

recognition models on mobile phones so that legit users can unlock the phones without depending on remote services or internet connections. The training stage may be done on a powerful server with significantly more computational resources than the inference stage, which is done on a mobile phone. Here, a key problem is how to use privileged provision, i.e., resources only accessible for training, to train a model with great inference performance [29].

Typical approaches to the problem are based on *knowledge distillation* (KD) [7, 9, 23]. As shown by the left half of Figure 2, KD consists of a *classifier* and a *teacher* [29]. To operate for resource-constrained inference, the classifier does not use privileged provision. On the other hand, the teacher uses privileged provision by, e.g., having a larger model capacity or taking more features as input. Once trained, the teacher outputs a distribution over labels called *soft labels* [29] for each training instance. Then, the teacher trains the classifier to predict the soft labels via a distillation loss such as the L2 loss on logits [7]. This training process is often called "distilling" the knowledge in the teacher into the classifier [23]. Since the teacher normally cannot perfectly model the true data distribution, it is difficult for the classifier to learn the true data distribution from the teacher.

Generative adversarial networks (GAN) provide an alternative way to learn the true data distribution. Inspired by Wang et al. [49], we first present a naive GAN (NaGAN) with two players. As shown by the right part of Figure 2, NaGAN consists of a classifier and a discriminator. The classifier serves as a generator that generates relevant labels given an instance while the discriminator aims to distinguish the true labels from the generated ones. The classifier learns from the discriminator to perfectly model the true data distribution at the equilibrium via adversarial losses. One limitation of NaGAN is that a large number of training instances and epochs is normally required to reach equilibrium [15], which restricts its applicability to domains where collecting labeled data is expensive. The slow training speed is because in such a two-player framework, the gradients from the discriminator to update the classifier often vanish or explode during the adversarial training [4]. It is challenging to train a classifier to learn the true data distribution with limited training instances and epochs.

To address this challenge, we propose a three-player framework named KDGAN to *distill knowledge with generative adversarial networks*. As shown in Figure 2, KDGAN consists of a *classifier*, a *teacher*, and a *discriminator*. In addition to the distillation loss in KD and the adversarial losses in NaGAN mentioned above, we define a distillation loss from the classifier to the teacher and an adversarial loss between the teacher and the discriminator. Specifically, the classifier and the teacher, serving as generators, aim to fool the discriminator by generating pseudo labels that resemble the true labels. Meanwhile, the classifier and the teacher try to reach an agreement on what pseudo labels to generate by distilling their knowledge into each other. By formulating the distillation and adversarial losses as a minimax game, we enable the classifier to learn the true data distribution at the equilibrium (see Section 3.2). Besides, the classifier receives gradients from the teacher via the distillation loss and the discriminator via the adversarial loss. The gradients from the teacher often have low variance, which reduces the variance of gradients and thus speeds up the adversarial training (see Section 3.3).

We further consider reducing the variance of the gradients from the discriminator to accelerate the training of KDGAN. The gradients from the discriminator may have large variance when obtained through the widely used policy gradient methods [49, 52]. It is non-trivial to obtain low-variance gradients from the discriminator because the classifier and the teacher generate discrete samples, which are not differentiable w.r.t. their parameters. We propose to relax the discrete distributions learned by the classifier and the teacher into concrete distributions [25, 31] with the Gumbel-Max trick [20, 30]. We use the concrete distributions for generating continuous samples to enable end-to-end differentiability and sufficient control over the variance of gradients. Given the continuous samples, we obtain low-variance gradients from the discriminator to accelerate the KDGAN training.

To summarize, our contributions are as follows:

- We propose a novel framework named KDGAN for multi-label learning, which trains a lightweight classifier suitable for resource-constrained inference using resources available only for training.
- We reduce the number of training epochs required to converge by decreasing the variance of gradients, which is achieved by the design of KDGAN and the Gumbel-Max trick.
- We conduct extensive experiments in two applications, image tag recommendation and deep model compression. The experiments validate the superiority of KDGAN over state-of-the-art methods.

2 Related Work

We briefly review studies on knowledge distillation (KD) and generative adversarial networks (GAN).

KD aims to transfer the knowledge in a powerful teacher to a lightweight classifier [9]. For example, Ba and Caruana [7] train a shallow classifier network to mimic a deep teacher network by matching logits via the L2 loss. Hinton et al. [23] generalize this work by training a classifier to predict soft labels provided by a teacher. Sau and Balasubramanian [39] further add random perturbations into soft labels to simulate learning from multiple teachers. Instead of using soft labels, Romero et al. [36] propose to use middle layers of a teacher to train a classifier. Unlike previous work on classification problems, Chen et al. [10] apply KD and hint learning to object detection problems. There also exists work that leverages KD to transfer knowledge between different domains [21], e.g., between high-quality and low-quality images [41]. Lopez-Paz et al. [29] unify KD with privileged information [35, 47, 48] as generalized distillation where a teacher is pretrained by taking as input privileged information. Compared to KD, the proposed KDGAN framework introduces a discriminator to guarantee that the classifier can learn the true data distribution at the equilibrium.

GAN is initially proposed to generate continuous data by training a generator and a discriminator adversarially in a minimax game [17]. GAN has only recently been introduced to generate discrete data [16, 54, 55] because discrete data makes it difficult to pass gradients from a discriminator backward to update a generator. For example, sequence GAN (SeqGAN) [52] models the process of token sequence generation as a stochastic policy and adopts Monte Carlo search to update a generator. Different from these GANs with two players, Li et al. propose a GAN with three players called Triple-GAN [13]. Our KDGAN also consists of three players including two generators and a discriminator, but differs from Triple-GAN in that: (1) Both generators in KDGAN learn a conditional distribution over labels given features. However, the generators in Triple-GAN learn a conditional distribution over labels given features and a conditional distribution over features given labels, respectively. (2) The samples from both generators in KDGAN are all discrete data while the samples from the generators in Triple-GAN include both discrete and continuous data. These differences lead to different objective functions and training techniques, e.g., KDGAN can use the Gumbel-Max trick [20, 30] to generate samples from both generators while Triple-GAN cannot do this. There is also a rich body of studies on improving the training of GAN [5, 33, 56] such as feature matching [38], which are orthogonal to our work and can be used to improve the training of KDGAN.

We explore the idea of integrating KD and GAN. A similar idea has been studied in [51] where a discriminator is introduced to train a classifier. This previous study [51] differs from ours in that their discriminator trains the classifier to learn the data distribution produced by the teacher, while our discriminator trains the classifier to learn the true data distribution.

We apply the proposed KDGAN to address the problem of deep model compression and image tag recommendation. We can also apply KDGAN to address the other problems where privileged provision is available [44]. For example, we can consider contextual signals in the intent tracking problem [42, 43] or user reviews in the movie recommendation problem [50] as privileged provision.

3 Methods

We study the problem of training a lightweight classifier from a teacher that is trained with privileged provision (denoted by ϱ) to satisfy stringent inference requirements. The inference requirements may include (1) running in real time with limited computational resources, where privileged provision is computational resources [23]; (2) lacking a certain type of input features, where privileged provision is privileged information [47]. Following existing work [29], we use multi-label learning problems [12, 18, 53] as the target application scenarios of our methods for illustration purpose.

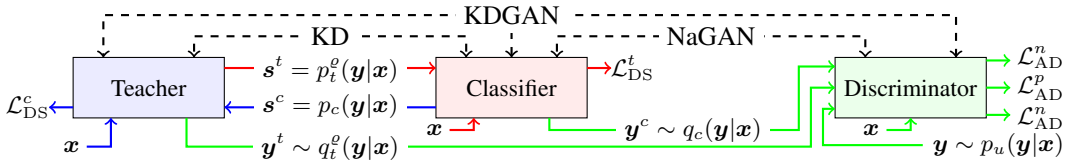


Figure 2: Comparison among KD, NaGAN, and KDGAN. The classifier (C) and the teacher (T) learn discrete categorical distributions $p_c(\boldsymbol{y}|\boldsymbol{x})$ and $p^\varrho_t(\boldsymbol{y}|\boldsymbol{x})$; $\boldsymbol{y}$ is a true label generated from the true data distribution $p_u(\boldsymbol{y}|\boldsymbol{x})$; $\boldsymbol{y}^c$ and $\boldsymbol{y}^t$ are continuous samples generated from concrete distributions $q_c(\boldsymbol{y}|\boldsymbol{x})$ and $q^\varrho_t(\boldsymbol{y}|\boldsymbol{x})$; $\boldsymbol{s}^c$ and $\boldsymbol{s}^t$ are soft labels produced by C and T; $\mathcal{L}^c_{\mathrm{DS}}$ and $\mathcal{L}^t_{\mathrm{DS}}$ are distillation losses for C and T; $\mathcal{L}^p_{\mathrm{AD}}$ and $\mathcal{L}^n_{\mathrm{AD}}$ are adversarial losses for positive and negative feature-label pairs.

Since privileged provision is only available at the training stage, the goal of the problem is to train a lightweight classifier that does not use privileged provision for effective inference.

To achieve this goal, we start with NaGAN, a naive adaptation of the two-player framework proposed by Wang et al. in information retrieval (Section 3.1). Similar to other two-player frameworks [49], the naive adaptation requires a large number of training instances and epochs [15], which is difficult to satisfy in practice [4]. To address the limitation, we propose a three-player framework named KDGAN that can speed up the training while preserving the equilibrium (Sections 3.2 and 3.3).

3.1 NaGAN Formulation

We begin with NaGAN that combines a classifier C with a discriminator D in a minimax game. Since D is not meant for inference, it can leverage privileged provision. For example, D may have a larger model capacity than C or take as input more features than those available to C. In NaGAN, C generates pseudo labels $\boldsymbol{y}$ given features $\boldsymbol{x}$ following a categorical distribution $p_c(\boldsymbol{y}|\boldsymbol{x})$, while D computes the probability $p^\varrho_d(\boldsymbol{x}, \boldsymbol{y})$ of a label $\boldsymbol{y}$ being from the true data distribution $p_u(\boldsymbol{y}|\boldsymbol{x})$ given features $\boldsymbol{x}$. With a slight abuse of notation, we also use $\boldsymbol{x}$ to refer to features including privileged information when the context is clear. Following the value function of IRGAN [49], we define the value function $V(c, d)$ for the minimax game in NaGAN as

$$\min_c \max_d V(c, d) = \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p^\varrho_d(\boldsymbol{x}, \boldsymbol{y})] + \mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1 - p^\varrho_d(\boldsymbol{x}, \boldsymbol{y}))]. \tag{1}$$

Let $h(\boldsymbol{x}, \boldsymbol{y})$ and $g(\boldsymbol{x}, \boldsymbol{y})$ be the scoring functions for C and D. We define $p_c(\boldsymbol{y}|\boldsymbol{x})$ and $p^\varrho_d(\boldsymbol{x}, \boldsymbol{y})$ as

$$p_c(\boldsymbol{y}|\boldsymbol{x}) = \mathrm{softmax}(h(\boldsymbol{x}, \boldsymbol{y})) \quad \text{and} \quad p^\varrho_d(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{sigmoid}(g(\boldsymbol{x}, \boldsymbol{y})). \tag{2}$$

The scoring functions can be implemented in various ways, e.g., $h(\boldsymbol{x}, \boldsymbol{y})$ can be a multilayer perceptron [27]. We will detail the scoring functions for specific applications in Section 4. Such a two-player framework is trained by updating C and D alternatively [49]. The training will proceed until the equilibrium is reached, where C learns the true data distribution. At that point, D can do no better than random guesses at deciding whether a given label is generated by C or not [6].

Our key observation is that the advantages and the disadvantages of KD and NaGAN are complementary: (1) KD usually requires a small number of training instances and epochs but cannot ensure the equilibrium where $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_u(\boldsymbol{y}|\boldsymbol{x})$. (2) NaGAN ensures the equilibrium where $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_u(\boldsymbol{y}|\boldsymbol{x})$ [49] but normally requires a large number of training instances and epochs. We aim to retain the advantages and avoid the disadvantages of both methods in a single framework.

3.2 KDGAN Formulation

We formulate KDGAN as a minimax game with a classifier C, a teacher T, and a discriminator D. Similar to the classifier C, the teacher T generates pseudo labels based on a categorical distribution $p^\varrho_t(\boldsymbol{y}|\boldsymbol{x}) = \mathrm{softmax}(f(\boldsymbol{x}, \boldsymbol{y}))$ where $f(\boldsymbol{x}, \boldsymbol{y})$ is also a scoring function. Both T and D use privileged provision, e.g., by having a large model capacity or taking privileged information as input. In KDGAN, D aims to maximize the probability of correctly distinguishing the true and pseudo labels, whereas C and T aim to minimize the probability that D rejects their generated pseudo labels. Meanwhile, C learns from T by mimicking the learned distribution of T. To build a general framework, we also enable T to learn from C because, in reality, a teacher's ability can also be enhanced by interacting

Algorithm 1: Minibatch stochastic gradient descent training of KDGAN.

Pretrain a classifier C, a teacher T, and a discriminator D with the training data $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$.
for *the number of training epochs* **do**
  for *the number of training steps for the discriminator* **do**
    Sample labels $\{\boldsymbol{y}_1, ..., \boldsymbol{y}_k\}$, $\{\boldsymbol{y}_1^c, ..., \boldsymbol{y}_k^c\}$, and $\{\boldsymbol{y}_1^t, ..., \boldsymbol{y}_k^t\}$ from $p_u(\boldsymbol{y}|\boldsymbol{x})$, $q_c(\boldsymbol{y}|\boldsymbol{x})$, and $q_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$.
    Update D by ascending along its gradients
    $\frac{1}{k}\sum_{i=1}^k \big(\nabla_d \log p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}_i) + \alpha\nabla_d \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^c)) + (1-\alpha)\nabla_d \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^t))\big)$.
  for *the number of training steps for the teacher* **do**
    Sample labels $\{\boldsymbol{y}_1^t, ..., \boldsymbol{y}_k^t\}$ from $q_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$ and update the teacher by descending along its gradients
    $\frac{1}{k}\sum_{i=1}^k (1-\alpha)\nabla_t \log q_t^\varrho(\boldsymbol{y}_i^t|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^t)) + \gamma\nabla_t \mathcal{L}_{\mathrm{DS}}^t(p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x}))$.
  for *the number of training steps for the classifier* **do**
    Sample labels $\{\boldsymbol{y}_1^c, ..., \boldsymbol{y}_k^c\}$ from $q_c(\boldsymbol{y}|\boldsymbol{x})$ and update C by descending along its gradients
    $\frac{1}{k}\sum_{i=1}^k \alpha\nabla_c \log q_c(\boldsymbol{y}_i^c|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^c)) + \beta\nabla_c \mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}))$.

with students (see Figure 6 in Appendix D for empirical evidence that T benefits from learning from C). Such a mutual learning helps C and T reduce their probability of generating different pseudo labels. Formally, we define the value function $U(c, t, d)$ for the minimax game in KDGAN as

$$\begin{aligned}&\min_{c,t}\max_d U(c,t,d) = \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d^\varrho(\boldsymbol{x},\boldsymbol{y})] + \alpha\mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))] \\ &+ (1-\alpha)\mathbb{E}_{\boldsymbol{y}\sim p_t^\varrho}[\log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))] + \beta\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})) + \gamma\mathcal{L}_{\mathrm{DS}}^t(p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})),\end{aligned} \tag{3}$$

where $\alpha \in (0, 1)$, $\beta \in (0, +\infty)$, and $\gamma \in (0, +\infty)$ are hyperparameters. We collectively refer to the expectation terms as the *adversarial losses* and refer to $\mathcal{L}_{\mathrm{DS}}^c$ and $\mathcal{L}_{\mathrm{DS}}^t$ as the *distillation losses*. The distillation losses can be defined in several ways [39], e.g., the L2 loss [7] or Kullback–Leibler divergence [23]. Note that $\mathcal{L}_{\mathrm{DS}}^c$ and $\mathcal{L}_{\mathrm{DS}}^t$ are used to train the classifier and the teacher, respectively.

Theoretical Analysis. We show that the classifier perfectly learns the true data distribution at the equilibrium of KDGAN. To see this, let $p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x}) = \alpha p_c(\boldsymbol{y}|\boldsymbol{x}) + (1-\alpha)p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$. It can be shown that the adversarial losses w.r.t. $p_c(\boldsymbol{y}|\boldsymbol{x})$ and $p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$ are equal to an adversarial loss w.r.t. $p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})$:

$$\begin{aligned}&\alpha\mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))] + (1-\alpha)\mathbb{E}_{\boldsymbol{y}\sim p_t^\varrho}[\log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))] \\ &= \alpha\textstyle\sum_{\boldsymbol{y}} p_c(\boldsymbol{y}|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) + (1-\alpha)\sum_{\boldsymbol{y}} p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) \\ &= \textstyle\sum_{\boldsymbol{y}} \big(\alpha p_c(\boldsymbol{y}|\boldsymbol{x}) + (1-\alpha)p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})\big) \log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) \\ &= \mathbb{E}_{\boldsymbol{y}\sim p_\alpha^\varrho}[\log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))].\end{aligned} \tag{4}$$

Therefore, let $\mathcal{L}_{\mathrm{MD}} = \beta\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})) + \gamma\mathcal{L}_{\mathrm{DS}}^t(p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x}))$ and $\mathcal{L}_{\mathrm{JS}}$ be the Jensen-Shannon divergence, the value function $U(c, t, d)$ of the minimax game can be rewritten as

$$\begin{aligned}&\min_\alpha\max_d \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d^\varrho(\boldsymbol{x},\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{y}\sim p_\alpha^\varrho}[\log(1 - p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))] + \mathcal{L}_{\mathrm{MD}} \\ &= \min_\alpha 2\mathcal{L}_{\mathrm{JS}}(p_u(\boldsymbol{y}|\boldsymbol{x})||p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})) + \beta\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})) + \gamma\mathcal{L}_{\mathrm{DS}}^t(p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})) - \log(4).\end{aligned} \tag{5}$$

Here, $\mathcal{L}_{\mathrm{JS}}$ reaches the minimum if and only if $p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x}) = p_u(\boldsymbol{y}|\boldsymbol{x})$ and $\mathcal{L}_{\mathrm{DS}}^c$ (or $\mathcal{L}_{\mathrm{DS}}^t$) reaches the minimum if and only if $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$. Hence, the KDGAN equilibrium is reached if and only if $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}) = p_u(\boldsymbol{y}|\boldsymbol{x})$ where the classifier learns the true data distribution. We summarize the above discussions in Lemma 4.1 (the necessary and sufficient conditions of maximizing the value function) and Theorem 4.2 (achieving the equilibrium), respectively (see Appendix A for proofs).

Lemma 4.1. *For any fixed classifier and teacher, the value function $U(c, t, d)$ is maximized if and only if the distribution of the discriminator is given by $p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}) = {}^{p_u(\boldsymbol{y}|\boldsymbol{x})}/_{(p_u(\boldsymbol{y}|\boldsymbol{x}) + p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x}))}$.*

Theorem 4.2. *The equilibrium of the minimax game $\min_{c,t}\max_d U(c, t, d)$ is achieved if and only if $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}) = p_u(\boldsymbol{y}|\boldsymbol{x})$. At that point, $U(c, t, d)$ reaches the value $-\log(4)$.*

3.3 KDGAN Training

In this section, we detail techniques for accelerating the training speed of KDGAN via reducing the number of training epochs needed. As discussed in earlier studies [8, 46], the training speed is closely related to the variance of gradients. Comparing with NaGAN, the KDGAN framework by design

can reduce the variance of gradients. This is because the high variance of a random variable can be reduced by a low-variance random variable (detailed in Lemma 4.3) and as we will discuss, T provides gradients of lower variance than D does. To reduce the variance of gradients from D and attain sufficient control over the variance, we further propose to obtain gradients from a continuous space by relaxing the discrete samples, i.e., pseudo labels, propagated between the classifier (or the teacher) and the discriminator into continuous samples with a reparameterization trick [25, 31].

First, we show how KDGAN reduces the variance of gradients. As discussed above, C only receives gradients $\nabla_c V$ from D in NaGAN while it receives gradients $\nabla_c U$ from both D and T in KDGAN:

$$\nabla_c V = \nabla_c \mathcal{L}^n_{\mathrm{AD}}, \quad \nabla_c U = \lambda \nabla_c \mathcal{L}^n_{\mathrm{AD}} + (1-\lambda)\nabla_c \mathcal{L}^c_{\mathrm{DS}}, \tag{6}$$

where $\lambda \in (0,1)$, $\nabla_c \mathcal{L}^n_{\mathrm{AD}}$ and $\nabla_c \mathcal{L}^c_{\mathrm{DS}}$ are gradients from D and T, respectively. Consistent with the findings in existing work [23, 39], we also observe that $\nabla_c \mathcal{L}^c_{\mathrm{DS}}$ usually has a lower variance than $\nabla_c \mathcal{L}^n_{\mathrm{AD}}$ (see Figure 7 in Appendix D for empirical evidence that the variance of $\nabla_c \mathcal{L}^c_{\mathrm{DS}}$ is smaller than that of $\nabla_c \mathcal{L}^n_{\mathrm{AD}}$ during the training process). Hence, it can be easily shown that the gradients w.r.t. C in KDGAN have a lower variance than that in NaGAN (refer to Lemma 4.3):

$$\mathrm{Var}(\nabla_c \mathcal{L}^c_{\mathrm{DS}}) \leq \mathrm{Var}(\nabla_c \mathcal{L}^n_{\mathrm{AD}}) \Rightarrow \mathrm{Var}(\nabla_c U) \leq \mathrm{Var}(\nabla_c V). \tag{7}$$

Next, we further reduce the variance of gradients with a reparameterization trick, in particular, the Gumbel-Max trick [20, 30]. The essence of the Gumbel-Max trick is to reparameterize generating discrete samples into a differentiable function of its parameters and an additional random variable of a Gumbel distribution. To perform the Gumbel-Max trick on generating discrete samples from the categorical distribution $p_c(\boldsymbol{y}|\boldsymbol{x})$, a concrete distribution [25, 31] can be used. We use a concrete distribution $q_c(\boldsymbol{y}|\boldsymbol{x})$ to generate continuous samples and use the continuous samples to compute the gradients $\nabla_c \mathcal{L}^n_{\mathrm{AD}}$ of the adversarial loss w.r.t. the classifier as

$$\nabla_c \mathcal{L}^n_{\mathrm{AD}} = \nabla_c \mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1 - p^o_d(\boldsymbol{x}, \boldsymbol{y}))] = \mathbb{E}_{\boldsymbol{y}\sim q_c}[\nabla_c \log q_c(\boldsymbol{y}|\boldsymbol{x}) \log(1 - p^o_d(\boldsymbol{x}, \boldsymbol{z}))]. \tag{8}$$

Here, $\boldsymbol{z} = \mathrm{onehot}(\mathrm{argmax}\, \boldsymbol{y})$ is a discrete pseudo label where $\boldsymbol{y} \sim q_c(\boldsymbol{y}|\boldsymbol{x})$. We define $q_c(\boldsymbol{y}|\boldsymbol{x})$ as

$$q_c(\boldsymbol{y}|\boldsymbol{x}) = \mathrm{softmax}(\frac{\log p_c(\boldsymbol{y}|\boldsymbol{x}) + \boldsymbol{g}}{\tau}), \quad \boldsymbol{g} \sim \mathrm{Gumbel}(0,1). \tag{9}$$

Here, $\tau \in (0, +\infty)$ is a temperature parameter and $\mathrm{Gumbel}(0,1)$ is the Gumbel distribution[2] [31]. We leverage the temperature parameter τ to control the variance of gradients over the training. With a high temperature, the samples from the concrete distribution are smooth, which give low-variance gradient estimates. Note that a disadvantage of the concrete distribution is that with a high temperature, it becomes a less accurate approximation to the original categorical distribution, which causes biased gradient estimates. We will discuss how to tune the temperature parameter in Section 4.

In addition to improving the training of C, we also apply the same techniques to improve the training of T. We update D with the back-propagation algorithm [37] (detailed in Appendix B). The overall logic of the KDGAN training is summarized in Algorithm 1. The three players can be first pretrained separately and then trained alternatively via minibatch stochastic gradient descent.

4 Experiments

The proposed KDGAN framework can be applied to a wide range of multi-label learning tasks where privileged provision is available. To show the applicability of KDGAN, we conduct experiments with the tasks of deep model compression (Section 4.1) and image tag recommendation (Section 4.2). Note that privileged provision is referred to as computational resources in deep model compression and privileged information in image tag recommendation, respectively.

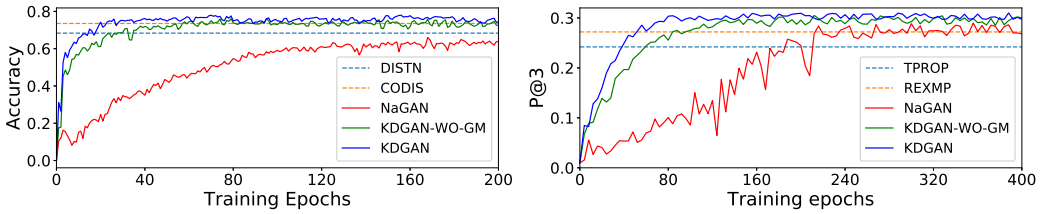
We implement KDGAN based on Tensorflow [1] and here we briefly describe our experimental setup[3]. We use two formulations of the distillation losses including the L2 loss [7] and the Kullback–Leibler divergence [23]. The two formulations exhibit comparable results and the results presented are based on the L2 loss [7]. Since both T and D can use privileged provision, we implement their scoring functions $f(\boldsymbol{x}, \boldsymbol{y})$ and $g(\boldsymbol{x}, \boldsymbol{y})$ using the same function $s(\boldsymbol{x}, \boldsymbol{y})$ but with different sets of parameters.

[2] The Gumbel distribution can be sampled by drawing $\boldsymbol{u} \sim \mathrm{Uniform}(0,1)$ and computing $\boldsymbol{g} = -\log(-\log \boldsymbol{u})$.

[3] The code and the data are made available at `https://github.com/xiaojiew1/KDGAN/`.

Table 1: Average accuracy over 10 runs in model compression (n is the number of training instances).

Method	MNIST			CIFAR-10		
	$n = 100$	$n = 1,000$	$n = 10,000$	$n = 500$	$n = 5,000$	$n = 50,000$
CODIS	74.02 ± 0.13	95.77 ± 0.10	98.89 ± 0.08	54.17 ± 0.20	77.82 ± 0.14	85.12 ± 0.11
DISTN	68.34 ± 0.06	93.97 ± 0.08	98.79 ± 0.07	50.92 ± 0.18	76.59 ± 0.15	83.32 ± 0.08
NOISY	66.53 ± 0.18	93.45 ± 0.11	98.58 ± 0.11	50.18 ± 0.28	75.42 ± 0.19	82.99 ± 0.12
MIMIC	67.35 ± 0.15	93.78 ± 0.13	98.65 ± 0.05	51.74 ± 0.23	75.66 ± 0.17	84.33 ± 0.10
NaGAN	64.90 ± 0.31	93.60 ± 0.22	98.95 ± 0.19	46.29 ± 0.32	76.11 ± 0.24	85.34 ± 0.27
KDGAN	$\mathbf{77.95} \pm 0.05$	$\mathbf{96.42} \pm 0.05$	$\mathbf{99.25} \pm 0.02$	$\mathbf{57.56} \pm 0.13$	$\mathbf{79.36} \pm 0.04$	$\mathbf{86.50} \pm 0.04$



(a) Deep model compression over MNIST.

(b) Image tag recommendation on YFCC100M.

Figure 3: Training curves of the classifier in the proposed NaGAN and KDGAN.

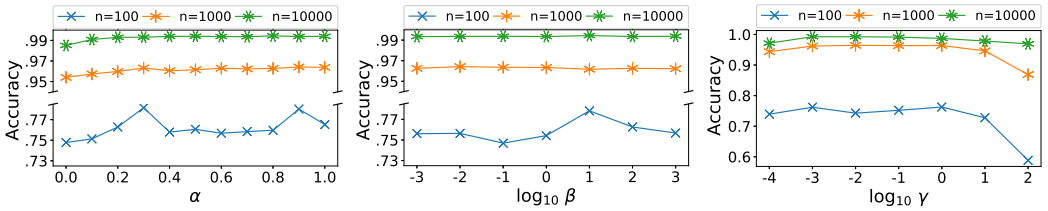
We search for the optimal values for the hyperparameters α in $[0.0, 1.0]$, β in $[0.001, 1000]$, and γ in $[0.0001, 100]$ based on validation performance. We find that a reasonable annealing schedule for the temperature parameter τ is to start with a large value (1.0) and exponentially decay it to a small value (0.1). We leave the exploration of the optimal schedule for future work.

4.1 Deep Model Compression

Deep model compression aims to reduce the storage and runtime complexity of deep models and to improve the deployability of such models on portable devices such as smart phones. Extensive computational resources available for training are considered privileged provision in this task.

Dataset and Setup. We use the widely adopted MNIST [27] and CIFAR-10 [26] datasets. The MNIST dataset has 60,000 grayscale images (50,000 for training and 10,000 for testing) with 10 different label classes. Following an earlier work [39], we do not preprocess the images on MNIST. The CIFAR-10 dataset has 60,000 colored images (50,000 for training and 10,000 for testing) with 10 different label classes. We preprocess the images by subtracting per-pixel mean, and we augment the training data by mirrored images. We vary the number of training instances in $[100, 10000]$ on MNIST and in $[500, 50000]$ on CIFAR-10. The scoring functions $h(\boldsymbol{x}, \boldsymbol{y})$ and $s(\boldsymbol{x}, \boldsymbol{y})$ are implemented as an MLP (1.2M parameters) and a LeNet (3.1M parameters) on MNIST; while $h(\boldsymbol{x}, \boldsymbol{y})$ and $s(\boldsymbol{x}, \boldsymbol{y})$ are implemented as a LeNet (0.5M parameters) and a ResNet (1.7M parameters) on CIFAR-10 (detailed in Appendix C). We evaluate various methods over 10 runs with different initialization of C and report the mean accuracy and the standard deviation. Since the focus of this paper is to achieve a better accuracy for a given architecture of the classifier, we defer the discussion on the classifier's ratio of compression and loss of accuracy w.r.t. the teacher to Table 3 in Appendix D.

Results and Discussions. First, we compare the proposed NaGAN and KDGAN with KD-based methods including MIMIC [7], DISTN [23], NOISY [39], and CODIS [2]. The results obtained by varying the number of training images on MNIST and CIFAR-10 are summarized in Table 1. On both datasets, KDGAN consistently outperforms the KD-based methods by a large margin. For example, KDGAN achieves as much as 5.31% performance gain with 100 training images on MNIST. We further compare NaGAN with the KD-based methods. We observe that NaGAN performs better when a large amount of training data are available (e.g., 50,000 training images on CIFAR-10) while KD-based methods perform better when a small number of training images are available (e.g., 500 training images on CIFAR-10). This is consistent with our analysis in Section 3.1 that NaGAN can learn the true data distribution better, although this requires a large amount of training data.



(a) Effect of varying α (b) Effect of varying β (c) Effect of varying γ

Figure 4: Effects of hyperparameters in KDGAN on MNIST for deep model compression.

Then, we compare NaGAN with KDGAN. As shown in Table 1, KDGAN achieves a larger performance gain over NaGAN with fewer training instances. This indicates that KDGAN requires a smaller number of training instances than NaGAN does to reach the same level of accuracy. This can be explained by that KDGAN introduces T to provide soft labels for training C. The soft labels generally have high entropy and reveal much useful information about each training instance. Hence, the soft labels impose much more constraint on the parameters of C than the true labels, which can reduce the number of training instances required to train C. We further investigate the training speed of NaGAN and KDGAN by the number of training epochs. Typical learning curves of C in NaGAN and KDGAN are shown in Figure 3a. Due to the page limit, we only show the results using 100 training images on MNIST. We find that KDGAN converges to a better accuracy with a smaller number of training epochs (about 25 epochs) than NaGAN (about 135 epochs). After convergence, the training curve in KDGAN is more stable than that in NaGAN. Moreover, we investigate the benefit provided by the Gumbel-Max trick for the KDGAN training. We perform the KDGAN training without using the Gumbel-Max trick (referred to as KDGAN-WO-GM) and also plot the accuracy against training epochs in Figure 3a. By comparing KDGAN with KDGAN-WO-GM, we can see that the Gumbel-Max trick speeds up the training process by around 45% in terms of training epochs. The Gumbel-Max trick also helps improve the accuracy from 0.7605 to 0.7795 (by around 2.5%). One possible reason is that the Gumbel-Max trick effectively reduces the gradient variance from the discriminator as discussed in Section 3.3. This is also observed in our experiments, e.g., by comparing the gradient variance from the adversarial loss not using the Gumbel-Max trick in Figure 7a with the one using the Gumbel-Max trick in Figure 7b (see Appendix D for details).

Next, we study the reasons for the higher accuracy of KDGAN. We present how the accuracy of KDGAN varies against the hyperparameters on the MNIST dataset in Figure 4 (Note the logarithmic scale of the x-axis in Figures 4b and 4c). We find that α and β have a relatively small effect on the accuracy, which suggests that KDGAN is a robust framework. Besides, if we set β to a small value (0.0001), we get more than 2% accuracy drop when KDGAN is trained with 100 training instances. This shows that T is important in training C when the number of training instances is small. We further find that a large value of γ causes the accuracy to deteriorate rapidly. This is because the soft labels provided by C are usually noisy. Emphasizing on training T to predict the noisy labels decreases the accuracy of T, which in turn decreases the accuracy of C. We obtain similar results for the effects of the hyperparameters on the CIFAR-10 dataset.

4.2 Image Tag Recommendation

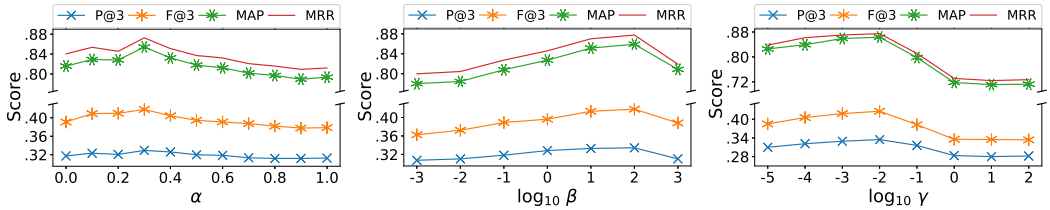
Image tag recommendation aims to recommend relevant tags (i.e., labels) after a user uploads an image to image-hosting websites such as Flickr[4]. As discussed before, we aim to recommend relevant tags right after a user uploads an image. This way, the user can just select from the recommended tags instead of inputting tags. Users may continue to add additional text for an uploaded image such as image titles and descriptions. We only use such additional text at the training stage as privileged information used by the teacher and the discriminator only. At the inference stage, our trained model (i.e., the classifier) only takes an image as input to make tag recommendations.

Dataset and Setup. We use the Yahoo Flickr Creative Commons 100 Million (YFCC100M) dataset[5] in the experiments [45]. To simulate the case where additional text about images is available for training, we randomly sample 20,000 images with titles or descriptions for training and another 2,000

[4] `https://www.flickr.com/`. [5] Yahoo Webscope Program. `http://webscope.sandbox.yahoo.com/`.

Table 2: Performance of various methods on the YFCC100M dataset in tag recommendation.

Method	Most Popular Tags						Randomly Sampled Tags					
	P@3	P@5	F@3	F@5	MAP	MRR	P@3	P@5	F@3	F@5	MAP	MRR
KNN	.2320	.1680	.2339	.1633	.5755	.5852	.1623	.1198	.1575	.1088	.3970	.4092
TPROP	.2420	.1636	.2811	.1949	.6177	.6270	.1883	.1372	.1810	.1252	.4512	.4636
TFEAT	.2560	.1752	.2871	.1999	.6417	.6503	.2002	.1420	.2195	.1495	.5149	.5309
REXMP	.2720	.1800	.3324	.2295	.7015	.7122	.2228	.1378	.2427	.1669	.5205	.5331
NaGAN	.2892	.1880	.3516	.2352	.7432	.7555	.2415	.1495	.2693	.1867	.5791	.5911
KDGAN	**.3047**	**.1968**	**.3678**	**.2526**	**.7787**	**.7905**	**.2572**	**.1666**	**.2946**	**.2009**	**.6302**	**.6452**



(a) Effect of varying α (b) Effect of varying β (c) Effect of varying γ

Figure 5: Effects of hyperparameters in KDGAN on YFCC100M for image tag recommendation.

images for testing. We create a dataset of images labeled with the 200 most popular tags and another dataset of images labeled with 200 randomly sampled tags. Following an earlier study [3], we use a VGGNet [40] pretrained on ImageNet [14] to extract image features and a LSTM [24] with pretrained word embeddings [34] to learn text features. We implement $h(\boldsymbol{x}, \boldsymbol{y})$ as an MLP with image features as input and implement $s(\boldsymbol{x}, \boldsymbol{y})$ as an MLP with the element-wise product of image and text features as input (detailed in Appendix C). We use precision (P@N), F-score (F@N), mean average precision (MAP), and mean reciprocal ranking (MRR) to evaluate performance.

Results and Discussions. First, we compare C in KDGAN with KNN [32], TPROP [19], TFEAT [11], and REXMP [28]. The overall results are presented in Table 2. We find that KDGAN achieves significant improvements over the other methods across all the measures. Although KDGAN does not explicitly model the semantic similarity between two labels like what REXMP does, it still makes better recommendations than REXMP does. The reason is that in KDGAN, T provides C with soft labels at training. The soft labels contain a rich similarity structure over tags which cannot be modeled well by any pairwise similarity between tags used in REXMP. For example, an image labeled with a tag `volleyball` is supplied with a soft label assigning a probability of 10^{-2} to `basketball`, 10^{-4} to `baseball`, and 10^{-8} to `dragonfly`. The reason that T generalizes is reflected in the relative probabilities over tags, which can be used for guiding C to generalize better.

Next, we compare the training curves of NaGAN, KDGAN-WO-GM, and KDGAN. We only plot the performance measured by P@3 in Figure 3b because the other measures exhibit similar training curves. We find that KDGAN learns a more accurate classifier with a smaller number of training epochs (about 100 epochs) than NaGAN (about 220 epochs) and KDGAN-WO-GM (about 150 epochs). After convergence, KDGAN consistently outperforms the best baseline REXMP.

Last, we investigate how the performance of KDGAN varies against the hyperparameters over the YFCC100M dataset. The results are summarized in Figure 5, which are consistent with our observations in the task of deep model compression.

5 Conclusion

We proposed a framework named KDGAN to distill knowledge with generative adversarial networks for multi-label learning with privileged provision. We have defined the KDGAN framework as a minimax game where a classifier, a teacher, and a discriminator are trained adversarially. We have proved that the minimax game has an equilibrium where the classifier perfectly models the true data distribution. We use the concrete distribution to control the variance of gradients during the

adversarial training and obtained low-variance gradient estimates to accelerate the training. We have shown that KDGAN outperforms the state-of-the-art methods in two important applications, image tag recommendation and deep model compression. We show that KDGAN learns a more accurate classifier at a faster speed than a naive GAN (NaGAN) does. For future work, we will explore adaptive methods for determining model hyperparameters to achieve better training dynamics.

Acknowledgement

This work is supported by Australian Research Council Future Fellowship Project FT120100832 and Discovery Project DP180102050. We thank the anonymous reviewers for their feedback on the paper. We have incorporated responses to reviewers' comments in the paper.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, 2016.

[2] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton. Large scale distributed neural network training through online distillation. In *ICLR*, 2018.

[3] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh. Vqa: Visual question answering. In *ICCV*, 2015.

[4] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *ICLR*, 2017.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[6] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang. Generalization and equilibrium in generative adversarial nets (gans). In *ICML*, 2017.

[7] J. Ba and R. Caruana. Do deep nets really need to be deep? In *NIPS*, 2014.

[8] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

[9] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *SIGKDD*, 2006.

[10] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker. Learning efficient object detection models with knowledge distillation. In *NIPS*, 2017.

[11] L. Chen, D. Xu, I. W. Tsang, and J. Luo. Tag-based image retrieval improved by augmented features and group-based refinement. *IEEE Transactions on Multimedia*, 2012.

[12] W. Cheng, E. Hüllermeier, and K. J. Dembczynski. Label ranking methods based on the plackett-luce model. In *ICML*, 2010.

[13] L. Chongxuan, T. Xu, J. Zhu, and B. Zhang. Triple generative adversarial nets. In *NIPS*, 2017.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[15] S. Feizi, C. Suh, F. Xia, and D. Tse. Understanding gans: the lqg setting. *arXiv preprint arXiv:1710.10793*, 2017.

[16] Z. Gan, L. Chen, W. Wang, Y. Pu, Y. Zhang, H. Liu, C. Li, and L. Carin. Triangle generative adversarial networks. In *NIPS*, 2017.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

[18] M. Grbovic, N. Djuric, S. Guo, and S. Vucetic. Supervised clustering of label ranking data using label preference information. *Machine learning*, 2013.

[19] M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *ICCV*, 2009.

[20] E. Gumbel. Statistical theory of extreme values and some practical applications: A series of lectures. *US Government Printing Office, Washington*, 1954.

[21] S. Gupta, J. Hoffman, and J. Malik. Cross modal distillation for supervision transfer. In *CVPR*, 2016.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[23] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS workshop*, 2014.

[24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[25] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[26] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[28] X. Li and C. G. Snoek. Classifying tag relevance with relevant positive and negative examples. In *ACMMM*, 2013.

[29] D. Lopez-Paz, L. Bottou, B. Schölkopf, and V. Vapnik. Unifying distillation and privileged information. In *ICLR*, 2016.

[30] C. J. Maddison, D. Tarlow, and T. Minka. A* sampling. In *NIPS*, 2014.

[31] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

[32] A. Makadia, V. Pavlovic, and S. Kumar. Baselines for image annotation. *IJCV*, 2010.

[33] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In *ICLR*, 2017.

[34] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[35] D. Pechyony and V. Vapnik. On the theory of learnining with privileged information. In *NIPS*, 2010.

[36] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[38] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, 2016.

[39] B. B. Sau and V. N. Balasubramanian. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650*, 2016.

[40] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[41] J.-C. Su and S. Maji. Cross quality distillation. *arXiv preprint arXiv:1604.00433*, 2016.

[42] Y. Sun, N. J. Yuan, Y. Wang, X. Xie, K. McDonald, and R. Zhang. Contextual intent tracking for personal assistants. In *SIGKDD*, 2016.

[43] Y. Sun, N. J. Yuan, X. Xie, K. McDonald, and R. Zhang. Collaborative nowcasting for contextual recommendation. In *WWW*, 2016.

[44] Y. Sun, N. J. Yuan, X. Xie, K. McDonald, and R. Zhang. Collaborative intent prediction with real-time contextual data. *TOIS*, 2017.

[45] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. Yfcc100m: the new data in multimedia research. *Communications of the ACM*, 2016.

[46] G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *NIPS*, 2017.

[47] V. Vapnik and R. Izmailov. Learning using privileged information: similarity control and knowledge transfer. *JMLR*, 2015.

[48] V. Vapnik and A. Vashist. A new learning paradigm: Learning using privileged information. *Neural networks*, 2009.

[49] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, 2017.

[50] X. Wang, J. Qi, K. Ramamohanarao, Y. Sun, B. Li, and R. Zhang. A joint optimization approach for personalized recommendation diversification. In *PAKDD*, 2018.

[51] Z. Xu, Y.-C. Hsu, and J. Huang. Learning loss for knowledge distillation with conditional adversarial networks. *arXiv preprint arXiv:1709.00513*, 2017.

[52] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2017.

[53] M.-L. Zhang and Z.-H. Zhou. A review on multi-label learning algorithms. *TKDE*, 2014.

[54] Y. Zhang, Z. Gan, and L. Carin. Generating text via adversarial training. In *NIPS workshop on Adversarial Training*, 2016.

[55] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin. Adversarial feature matching for text generation. In *ICML*, 2017.

[56] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. In *ICLR*, 2017.

A Theoretical Analysis

In this section, we provide detailed proofs of the theoretical results in the KDGAN framework. Let $p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x}) = \alpha p_c(\boldsymbol{y}|\boldsymbol{x}) + (1-\alpha)p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$, which is referred to as the mixture distribution. We first show that the optimal distribution of the discriminator balances between the true data distribution $p_u(\boldsymbol{y}|\boldsymbol{x})$ and the mixture distribution $p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})$, as stated below.

Lemma 4.1. *For any fixed classifier and teacher, the value function $U(c,t,d)$ is maximized if and only if the distribution of the discriminator is given by $p_d^\varrho(\boldsymbol{x},\boldsymbol{y}) = {p_u(\boldsymbol{y}|\boldsymbol{x})}/{(p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x}))}$.*

Proof. Given the classifier $p_c(\boldsymbol{y}|\boldsymbol{x})$ and the teacher $p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$, the discriminator aims to maximize the value function $U(c,t,d)$ of the minimax game as

$$\begin{aligned}
&\max_d U(c,t,d) \\
&= \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d^\varrho(\boldsymbol{x},\boldsymbol{y})] + \alpha\mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1-p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))] + (1-\alpha)\mathbb{E}_{\boldsymbol{y}\sim p_t^\varrho}[\log(1-p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))] \\
&= \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d^\varrho(\boldsymbol{x},\boldsymbol{y})] + \alpha\sum\nolimits_{\boldsymbol{y}} p_c(\boldsymbol{y}|\boldsymbol{x})\log(1-p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) + (1-\alpha)\sum\nolimits_{\boldsymbol{y}} p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})\log(1-p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) \\
&= \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d^\varrho(\boldsymbol{x},\boldsymbol{y})] + \sum\nolimits_{\boldsymbol{y}} \big(\alpha p_c(\boldsymbol{y}|\boldsymbol{x}) + (1-\alpha)p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})\big)\log(1-p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) \\
&= \mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d^\varrho(\boldsymbol{x},\boldsymbol{y})] + \sum\nolimits_{\boldsymbol{y}} p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})\log(1-p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) \\
&= \sum\nolimits_{\boldsymbol{y}} p_u(\boldsymbol{y}|\boldsymbol{x})\log p_d^\varrho(\boldsymbol{x},\boldsymbol{y}) + \sum\nolimits_{\boldsymbol{y}} p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})\log(1-p_d^\varrho(\boldsymbol{x},\boldsymbol{y})) \\
&= F(p_d^\varrho(\boldsymbol{x},\boldsymbol{y})).
\end{aligned}$$

The function $F(p_d^\varrho(\boldsymbol{x},\boldsymbol{y}))$ achieves the maximum if and only if the distribution of the discriminator is equivalent to $p_d^\varrho(\boldsymbol{x},\boldsymbol{y}) = {p_u(\boldsymbol{y}|\boldsymbol{x})}/{p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})}$, completing the proof. □

Next, we show that the equilibrium of the minimax game is achieved if and only if both the classifier and the teacher perfectly model the true data distribution, which is summarized as follows.

Theorem 4.2. *The equilibrium of the minimax game $\min_{c,t}\max_d U(c,t,d)$ is achieved if and only if $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}) = p_u(\boldsymbol{y}|\boldsymbol{x})$. At that point, $U(c,t,d)$ reaches the value $-\log(4)$.*

Proof. Let $\mathcal{L}_{\mathrm{MD}} = \beta\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})) + \gamma\mathcal{L}_{\mathrm{DS}}^t(p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x}))$. Given the optimal distribution of the discriminator in Lemma 4.1, the classifier and the teacher aim to minimize the value function $U(c,t,d)$ of the minimax game as follows,

$$\begin{aligned}
&\min_{s,t} U(c,t,d) \\
&= \sum\nolimits_{\boldsymbol{y}} p_u(\boldsymbol{y}|\boldsymbol{x})\log\frac{p_u(\boldsymbol{y}|\boldsymbol{x})}{p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})} + \sum\nolimits_{\boldsymbol{y}} p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})\log(1-\frac{p_u(\boldsymbol{y}|\boldsymbol{x})}{p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})}) + \mathcal{L}_{\mathrm{MD}} \\
&= \sum\nolimits_{\boldsymbol{y}} p_u(\boldsymbol{y}|\boldsymbol{x})\log\frac{p_u(\boldsymbol{y}|\boldsymbol{x})}{p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})} + \sum\nolimits_{\boldsymbol{y}} p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})\log\frac{p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})}{p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})} + \mathcal{L}_{\mathrm{MD}} \\
&= -\log(4) + \mathcal{L}_{\mathrm{KL}}(p_u(\boldsymbol{y}|\boldsymbol{x})||\frac{p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})}{2}) + \mathcal{L}_{\mathrm{KL}}(p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})||\frac{p_u(\boldsymbol{y}|\boldsymbol{x})+p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})}{2}) + \mathcal{L}_{\mathrm{MD}} \\
&= -\log(4) + 2\mathcal{L}_{\mathrm{JS}}(p_u(\boldsymbol{y}|\boldsymbol{x})||p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})) + \beta\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})) + \gamma\mathcal{L}_{\mathrm{DS}}^t(p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})).
\end{aligned}$$

Here, $\mathcal{L}_{\mathrm{KL}}$ is the Kullback–Leibler divergence. $\mathcal{L}_{\mathrm{JS}}$ is the Jensen-Shannon divergence which is non-negative and reaches zero if and only if $p_u(\boldsymbol{y}|\boldsymbol{x}) = p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x})$. The distillation losses $\mathcal{L}_{\mathrm{DS}}^c$ and $\mathcal{L}_{\mathrm{DS}}^t$ such as the L2 loss on logits [7] and the Kullback–Leibler divergence on distributions [23] achieve the minimum at zero if and only if $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$. Therefore, the value function $U(c,t,d)$ reaches the minimum at $-\log(4)$ if and only if $p_c(\boldsymbol{y}|\boldsymbol{x}) = p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}) = p_\alpha^\varrho(\boldsymbol{y}|\boldsymbol{x}) = p_u(\boldsymbol{y}|\boldsymbol{x})$, which completes the proof. □

Further, we show that the high variance of a random variance can be reduced with a low-variance random variance, which is summarized in Lemma 4.3.

Lemma 4.3. *Let X and Y be random variables with $\mathrm{Var}(X) \leq \mathrm{Var}(Y)$. Let $Z = \lambda X + (1-\lambda)Y$, then we have $\mathrm{Var}(Z) \leq \mathrm{Var}(Y)$ for all $\lambda \in (0,1)$.*

Proof. Given $\mathrm{Var}(X) \leq \mathrm{Var}(Y)$, the covariance $\mathrm{Cov}(X, Y)$ is less than or equal to $\mathrm{Var}(Y)$ because

$$\mathrm{Cov}(X, Y) \leq |\,\mathrm{Cov}(X, Y)| \leq \sqrt{\mathrm{Var}(X)\,\mathrm{Var}(Y)} \leq \sqrt{\mathrm{Var}(Y)\,\mathrm{Var}(Y)} \leq \mathrm{Var}(Y).$$

According to the properties of the variance, for all $\lambda \in (0, 1)$, we have

$$\begin{aligned}\mathrm{Var}(Z) &= \lambda^2\,\mathrm{Var}(X) + 2\lambda(1-\lambda)\,\mathrm{Cov}(X, Y) + (1-\lambda)^2\,\mathrm{Var}(Y)\\ &\leq \lambda^2\,\mathrm{Var}(Y) + 2\lambda(1-\lambda)\,\mathrm{Cov}(X, Y) + (1-\lambda)^2\,\mathrm{Var}(Y)\\ &\leq \lambda^2\,\mathrm{Var}(Y) + 2\lambda(1-\lambda)\,\mathrm{Var}(Y) + (1-\lambda)^2\,\mathrm{Var}(Y)\\ &= \mathrm{Var}(Y),\end{aligned}$$

This completes the proof. □

B Gradient Derivation

We provide detailed derivations of the gradient computation in the KDGAN framework. Similar to the definition of the concrete distribution $q_c(\boldsymbol{y}|\boldsymbol{x})$ for the classifier in Equation 9, we first define a concrete distribution $q_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$ for the teacher as follows,

$$q_t^\varrho(\boldsymbol{y}|\boldsymbol{x}) = \mathrm{softmax}(\frac{\log p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}) + \boldsymbol{g}}{\tau}), \quad \boldsymbol{g} \sim \mathrm{Gumbel}(0, 1),$$

where $\tau \in (0, +\infty)$ is a temperature parameter and $\mathrm{Gumbel}(0, 1)$ is the Gumbel distribution [31]. The classifier and the teacher generate continuous samples from the concrete distributions $q_c(\boldsymbol{y}|\boldsymbol{x})$ and $q_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$, respectively, and then discretize the continuous samples into pseudo labels. The discriminator aims to maximize the probability of correctly identifying the true labels as positive and the pseudo labels as negative. The discriminator is trained to maximize the value function $U(c, t, d)$ of the minimax game by ascending along its gradients

$$\begin{aligned}&\nabla_d U(c, t, d)\\ &= \nabla_d\big(\mathbb{E}_{\boldsymbol{y}\sim p_u}[\log p_d^\varrho(\boldsymbol{x}, \boldsymbol{y})] + \alpha\mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}))] + (1-\alpha)\mathbb{E}_{\boldsymbol{y}\sim p_t^\varrho}[\log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}))]\big)\\ &\approx \frac{1}{k}\sum\nolimits_{i=1}^{k}\big(\nabla_d \log p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}_i) + \alpha\nabla_d \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^c)) + (1-\alpha)\nabla_d \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^t))\big).\end{aligned}$$

Here, k is the number of samples used to estimate the gradients. The true label $\boldsymbol{y}_i$ is sampled from the true data distribution $p_u(\boldsymbol{y}|\boldsymbol{x})$. $\boldsymbol{z}_i^c = \mathrm{onehot}(\mathrm{argmax}\,\boldsymbol{y}_i^c)$ and $\boldsymbol{z}_i^t = \mathrm{onehot}(\mathrm{argmax}\,\boldsymbol{y}_i^t)$ are pseudo labels where $\boldsymbol{y}_i^c \sim q_c(\boldsymbol{y}|\boldsymbol{x})$ and $\boldsymbol{y}_i^t \sim q_t^\varrho(\boldsymbol{y}|\boldsymbol{x})$ are continuous samples.

The classifier aims to generate the pseudo labels that resemble the true labels and predict the soft labels produced by the teacher. The classifier is trained to minimize the value function $U(c, t, d)$ of the minimax game by descending along its gradients

$$\begin{aligned}\nabla_c U(c, t, d) &= \nabla_c\big(\alpha\mathbb{E}_{\boldsymbol{y}\sim p_c}[\log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}))] + \beta\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}))\big)\\ &= \alpha\nabla_c \sum\nolimits_{\boldsymbol{y}} p_c(\boldsymbol{y}|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y})) + \beta\nabla_c\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}))\\ &= \alpha\sum\nolimits_{\boldsymbol{y}} \nabla_c p_c(\boldsymbol{y}|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y})) + \beta\nabla_c\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}))\\ &= \alpha\sum\nolimits_{\boldsymbol{y}} p_c(\boldsymbol{y}|\boldsymbol{x})\nabla_c \log p_c(\boldsymbol{y}|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y})) + \beta\nabla_c\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}))\\ &= \alpha\mathbb{E}_{\boldsymbol{y}\sim p_c}[\nabla_c \log p_c(\boldsymbol{y}|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}))] + \beta\nabla_c\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x}))\\ &\approx \frac{\alpha}{k}\sum\nolimits_{i=1}^{k} \nabla_c \log q_c(\boldsymbol{y}_i^c|\boldsymbol{x}) \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^c)) + \beta\nabla_c\mathcal{L}_{\mathrm{DS}}^c(p_c(\boldsymbol{y}|\boldsymbol{x}), p_t^\varrho(\boldsymbol{y}|\boldsymbol{x})),\end{aligned}$$

where $\boldsymbol{z}_i^c = \mathrm{onehot}(\mathrm{argmax}\,\boldsymbol{y}_i^c)$ is a pseudo label and $\boldsymbol{y}_i^c \sim q_c(\boldsymbol{y}|\boldsymbol{x})$ is a continuous sample. At the training of the classifier, we use a control variate [49], which is defined as

$$b_c = \mathbb{E}_{\boldsymbol{y}\sim p_c(\boldsymbol{y}|\boldsymbol{x})}[\log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{y}))] \approx \sum\nolimits_{i=1}^{k} \log(1 - p_d^\varrho(\boldsymbol{x}, \boldsymbol{z}_i^c)),$$

where $\boldsymbol{z}_i^c = \mathrm{onehot}(\mathrm{argmax}\,\boldsymbol{y}_i^c)$ is obtained by discretizing a continuous sample $\boldsymbol{y}_i^c \sim q_c(\boldsymbol{y}|\boldsymbol{x})$. $\nabla_c\mathcal{L}_{\mathrm{DS}}^c$ is the gradients of the distillation loss $\mathcal{L}_{\mathrm{DS}}^c$ w.r.t. the classifier, which can be easily computed

by the back-propagation algorithm. For example, if we use the L2 loss on logits [7] to define the distillation loss $\mathcal{L}^c_{\mathrm{DS}}$ as

$$\mathcal{L}^c_{\mathrm{DS}}(p_c(\boldsymbol{y}|\boldsymbol{x}), p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x})) = \frac{1}{2}||\log p_c(\boldsymbol{y}|\boldsymbol{x})) - \log p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x})||^2_2,$$

the gradients $\nabla_c \mathcal{L}^c_{\mathrm{DS}}$ are computed by

$$\nabla_c \mathcal{L}^c_{\mathrm{DS}}(p_c(\boldsymbol{y}|\boldsymbol{x}), p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x})) = ||\log p_c(\boldsymbol{y}|\boldsymbol{x})) - \log p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x})||_2 \nabla_c \log p_c(\boldsymbol{y}|\boldsymbol{x}).$$

Similarly, the gradients to update the teacher are derived as follows,

$$\begin{aligned}
\nabla_t U(c,t,d) &= \nabla_t\big((1-\alpha)\mathbb{E}_{\boldsymbol{y}\sim p^{\varrho}_t}[\log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{y}))] + \gamma\mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x}))\big) \\
&= (1-\alpha)\sum\nolimits_{\boldsymbol{y}} \nabla_t p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}) \log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{y})) + \gamma\nabla_t\mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})) \\
&= (1-\alpha)\sum\nolimits_{\boldsymbol{y}} \nabla_t p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}) \log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{y})) + \gamma\nabla_t\mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})) \\
&= (1-\alpha)\sum\nolimits_{\boldsymbol{y}} p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}) \nabla_t \log p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}) \log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{y})) + \gamma\nabla_t\mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})) \\
&= (1-\alpha)\mathbb{E}_{\boldsymbol{y}\sim p^{\varrho}_t}[\nabla_t \log p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}) \log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{y}))] + \gamma\nabla_t\mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})) \\
&\approx \frac{1-\alpha}{k}\sum\nolimits_{i=1}^{k} \nabla_t \log q^{\varrho}_t(\boldsymbol{y}^t_i|\boldsymbol{x}) \log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{z}^t_i)) + \gamma\nabla_t\mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})),
\end{aligned}$$

where $\boldsymbol{z}^t_i = \mathrm{onehot}(\mathrm{argmax}\, \boldsymbol{y}^t_i)$ is a pseudo label and $\boldsymbol{y}^t_i \sim q^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x})$ is a continuous sample. At the training of the teacher, we also use a control variate [49], which is defined as

$$b_t = \mathbb{E}_{\boldsymbol{y}\sim p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x})}[\log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{y}))] \approx \sum\nolimits_{i=1}^{k} \log(1-p^{\varrho}_d(\boldsymbol{x},\boldsymbol{z}^t_i)),$$

where $\boldsymbol{z}^t_i = \mathrm{onehot}(\mathrm{argmax}\, \boldsymbol{y}^t_i)$ is obtained by discretizing a continuous sample $\boldsymbol{y}^t_i \sim q^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x})$. $\nabla_t\mathcal{L}^t_{\mathrm{DS}}$ is the gradients of the distillation loss $\mathcal{L}^t_{\mathrm{DS}}$ w.r.t. the teacher. For example, the gradients $\nabla_t\mathcal{L}^t_{\mathrm{DS}}$ are given by

$$\nabla_t \mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})) = ||\log p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}) - \log p_c(\boldsymbol{y}|\boldsymbol{x})||_2 \nabla_t \log p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}),$$

when the distillation loss $\mathcal{L}^t_{\mathrm{DS}}$ is defined as the L2 loss on logits [7],

$$\mathcal{L}^t_{\mathrm{DS}}(p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}), p_c(\boldsymbol{y}|\boldsymbol{x})) = \frac{1}{2}||\log p^{\varrho}_t(\boldsymbol{y}|\boldsymbol{x}) - \log p_c(\boldsymbol{y}|\boldsymbol{x}))||^2_2.$$

C Network Architectures

We describe network architectures which we use to conduct experiments in deep model compression and image tag recommendation tasks. First, we describe the network architectures in deep model compression task on the MNIST dataset. We implement the scoring function $h(\boldsymbol{x}, \boldsymbol{y})$ as an MLP [27]. The architecture of the MLP is given by

1. An input layer of a 28×28 grayscale image.
2. A stack of 2 fully connected layers with 800 neurons.
3. A softmax layer with 10 classes.

We implement the scoring function $s(\boldsymbol{x}, \boldsymbol{y})$ as a LeNet [27]. The architecture of the LeNet is given by

1. An input layer of a 28×28 grayscale image.
2. A convolutional layer with 32 kernels of size 5×5 and stride 1.
3. A max pooling layer with size 2×2 and stride 2.
4. A convolutional layer with 64 kernels of size5×5 and stride 1.
5. A max pooling layer with size 2×2 and stride 2.
6. A fully connected layer with 1024 neurons.
7. A softmax layer with 10 classes.

Next, we describe the network architectures in deep model compression task on the CIFAR-10 dataset. We implement $h(\boldsymbol{x}, \boldsymbol{y})$ as a LeNet [27]. The architecture of the LeNet is given by

1. An input layer of a 32×32 colored image.
2. A convolutional layer with 64 kernels of size 5×5 and stride 1.
3. A max pooling layer with size 2×2 and stride 2.
4. A convolutional layer with 128 kernels of size 5×5 and stride 1.
5. A max pooling layer with size 2×2 and stride 2.
6. A fully connected layer with 1024 neurons.
7. A softmax layer with 10 classes.

We implement $s(\boldsymbol{x}, \boldsymbol{y})$ as a 101-layer ResNet [22]. The architecture of the ResNet is given by

1. An input layer of a 32×32 colored image.
2. A convolutional layer with 16 kernels with size 3×3 and stride 1.
3. Three stacked blocks of 3 convolutional layers which use 64 kernels of size 1×1, 64 kernels of size 3×3, and 256 kernels of size 1×1, respectively.
4. Four stacked blocks of 3 convolutional layers which use 128 kernels of size 1×1, 128 kernels of size 3×3, and 512 kernels of size 1×1, respectively.
5. Twenty three stacked blocks of 3 convolutional layers which use 256 kernels of size 1×1, 256 kernels of size 3×3, and 1024 kernels of size 1×1, respectively.
6. Three stacked blocks of 3 convolutional layers which use 512 kernels of size 1×1, 512 kernels of size 3×3, and 2048 kernels of size 1×1, respectively.
7. A global pooling layer.
8. A softmax layer with 10 classes.

Finally, we describe the network architectures in image tag recommendation task on the YFCC100M dataset. We use the same network architectures when experimenting with the two datasets of images labeled with the 200 most popular tags and 200 randomly sampled tags, respectively. We implement a VGGNet [40] to extract image features. The architecture of the VGGNet is written as

1. An input layer of a 224×224 colored image.
2. A stack of 2 convolutional layers with 64 kernels of size 3×3 and stride 1.
3. A max pooling layer with size 2×2 and stride 2.
4. A stack of 2 convolutional layers with 128 kernels of size 3×3 and stride 1.
5. A max pooling layer with size 2×2 and stride 2.
6. A stack of 2 convolutional layers with 256 kernels of size 3×3 and stride 1.
7. A max pooling layer with size 2×2 and stride 2.
8. A stack of 2 convolutional layers with 512 kernels of size 3×3 and stride 1.
9. A max pooling layer with size 2×2 and stride 2.
10. A stack of 2 convolutional layers with 512 kernels of size 3×3 and stride 1.
11. A max pooling layer with size 2×2 and stride 2.
12. A fully connected layer with 4096 neurons.
13. A fully connected layer with 4096 neurons.
14. A fully connected layer with 100 neurons.

We implement a LSTM [24] to learn text features. The architecture of the LSTM is written as

$$
\begin{aligned}
\boldsymbol{f}_t &= \mathrm{sigmoid}(\mathbf{W}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f), \\
\boldsymbol{i}_t &= \mathrm{sigmoid}(\mathbf{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i), \\
\boldsymbol{o}_t &= \mathrm{sigmoid}(\mathbf{W}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o), \\
\boldsymbol{s}_t &= \boldsymbol{f}_t \odot \boldsymbol{s}_{t-1} + \boldsymbol{i}_t \odot \tanh(\mathbf{W}_s \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_s), \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{s}_t),
\end{aligned}
$$

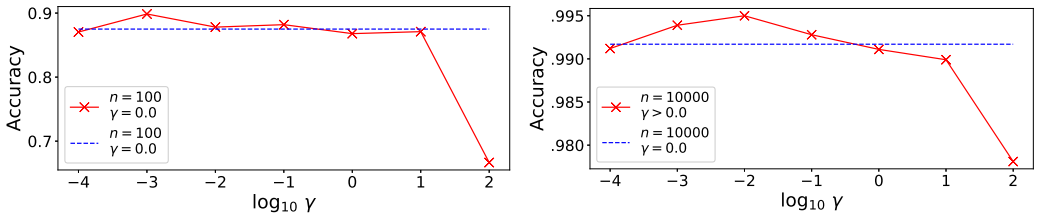
where $[\boldsymbol{h}, \boldsymbol{x}]$ is the vector concatenation and $\odot$ is the element-wise product. We set the hidden size of the LSTM to 100 in the experiments. Let $\boldsymbol{v}_x \in \mathbb{R}^{100}$ be an image feature vector extracted by the VGGNet and $\boldsymbol{v}_z \in \mathbb{R}^{100}$ be a text feature vector learned by the LSTM. We implement the scoring function $h(\boldsymbol{x}, \boldsymbol{y})$ as an MLP [3]. The architecture of the MLP is written as

1. An input layer of a feature vector with size 100 (i.e. the image features $\boldsymbol{v}_x$).
2. A stack of 2 fully connected layers with 800 neurons.
3. A softmax layer with 200 classes.

We implement the scoring function $s(\boldsymbol{x}, \boldsymbol{y})$ as an MLP [3]. The architecture of the MLP is given by

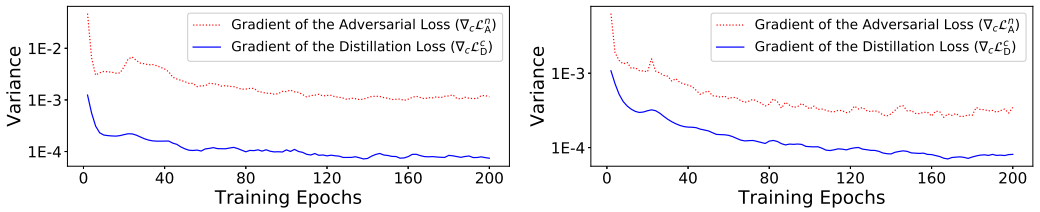
1. An input layer of a feature vector with size 100 (i.e. the element-wise product of $\boldsymbol{v}_x$ and $\boldsymbol{v}_z$).
2. A stack of 2 fully connected layers with 1200 neurons.
3. A softmax layer with 200 classes.

D Additional Experiments



(a) Using 100 training images. (b) Using 10,000 training images.

Figure 6: The accuracy of the teacher against the hyperparameter γ in KDGAN on MNIST. Note that γ controls how much the classifier distills its knowledge into the teacher.



(a) KDGAN without the Gumbel-Max trick. (b) KDGAN with Gumbel-Max trick.

Figure 7: Variances of the gradient of the adversarial loss ($\nabla_c \mathcal{L}_{\mathrm{AD}}^n$) or the distillation loss ($\nabla_c \mathcal{L}_{\mathrm{DS}}^c$) w.r.t. the classifier. The results are obtained by training KDGAN with 100 training images on MNIST.

We study the classifier's ratio of compression (number of parameters) and loss of accuracy w.r.t. the teacher on MNIST. We vary the hidden layer size of the classifier from 100 (89,610 parameters) to 1,200 (2,395,210 parameters) and present the results in Table 3 . The loss of accuracy generally decreases as the hidden layer size of the classifier or the number of training examples increases.

Table 3: Model size and accuracy of the classifier and the teacher (shown in parenthesis) in KDGAN on MNIST.

#Param. (M)	$n = 5$K	$n = 10$K	$n = 50$K
0.09 (3.12)	97.96 (99.25)	98.74 (99.42)	99.03 (99.65)
0.19 (3.12)	98.72 (99.26)	98.92 (99.46)	99.27 (99.70)
1.22 (3.12)	99.01 (99.28)	99.25 (99.48)	99.54 (99.72)
2.28 (3.12)	99.04 (99.27)	99.40 (99.53)	**99.77** (**99.78**)

Table 4: Average accuracy over 10 runs with varying training size (n) on MNIST.

Method	$n = 5$K	$n = 10$K	$n = 50$K
CODIS	98.53	98.89	99.31
DISTN	98.04	98.79	99.26
MIMIC	97.93	98.65	99.05
KDGAN	**99.01**	**99.25**	**99.54**