Learning Loss Functions for Semi-supervised Learning via Discriminative Adversarial Networks

Cicero Nogueira dos Santos*
IBM Research AI
Yorktown Heights, NY

Kahini Wadhawan
IBM Research AI
Yorktown Heights, NY

Bowen Zhou
JD.com
Beijing, China

Abstract

We propose discriminative adversarial networks (DAN) for semi-supervised learning and loss function learning. Our DAN approach builds upon generative adversarial networks (GANs) and conditional GANs but includes the key differentiator of using two discriminators instead of a generator and a discriminator. DAN can be seen as a framework to learn loss functions for predictors that also implements semi-supervised learning in a straightforward manner. We propose instantiations of DAN for two different prediction tasks: classification and ranking. Our experimental results on three datasets of different tasks demonstrate that DAN is a promising framework for both semi-supervised learning and learning loss functions for predictors. For all tasks, the semi-supervised capability of DAN can significantly boost the predictor performance for small labeled sets with minor architecture changes across tasks. Moreover, the loss functions automatically learned by DANs are very competitive and usually outperform the standard pairwise and negative log-likelihood loss functions for semi-supervised learning.

1 Introduction

One of the challenges in developing semi-supervised learning (SSL) algorithms is to define a loss (cost) function that handles both labeled and unlabeled data. Many SSL methods work by changing the original loss function to include an additional term that deals with the unlabeled data [32, 20, 23]. Recent advances in generative models have allowed the development of successful approaches that perform SSL while doing data generation, which allows the use of unlabeled data in more flexible ways. The two main families of successful generative approaches are based on variational autoencoders (VAE) [13] and generative adversarial networks [7]. Most GAN-based SSL approaches change the loss function of the discriminator to combine a supervised loss (e.g. negative log likelihood with respect to the ground truth labels) with the unsupervised loss normally used in the discriminator [26]. While VAE-based SSL approaches have achieved good results for tasks in both computer vision [14, 16] and natural language processing (NLP) domains [28, 30], GAN-based SSL have primarily targeted tasks from the computer vision domain [24, 22, 5, 15]. The main reason is that applying GANs to discrete data generation problems, e.g. natural language generation, is difficult because the generator network in GAN is designed to be able to adjust the output continuously, which does not (naturally) work on discrete data generation.

In this paper, we propose discriminative adversarial networks (DAN) for SSL and loss function learning. DAN builds upon GAN and conditional GAN but includes the key differentiator of using two discriminators instead of a generator and a discriminator. The first discriminator (the *predictor* P) produces the prediction y given a data point x, and the second discriminator (the *judge* J) takes in a pair (x, y) and judges if it is a *predicted label* pair or *human labeled* pair. While GAN can be seen as a method that implicitly learns loss functions for generative models, DAN can be seen as a method

*Corresponding author. Email: cicerons@us.ibm.com

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

that learns loss functions for predictors. The main benefits of DAN are: (1) The predictor P does not use information from labels, therefore unlabeled data can be used in a transparent way; (2) We do not need to manually define a loss function that handles both labeled and unlabeled data, the judge J implicitly learns the loss function used to optimize P; (3) Different from VAE and GAN-based SSL approaches, in DAN we do not have to perform data generation. This allows the application of SSL using adversarial networks for NLP sidestepping troubled discrete data generation; (4) Prediction problems with complex/structured outputs can benefit from DAN's implicit loss function learning capability. This is important because for many structured prediction tasks such as ranking and coreference resolution, researchers normally use surrogate loss functions since the best loss function for the task is too expensive to compute or, in some cases, because a good loss function is not known.

We have applied DAN for two NLP tasks: answer selection and text classification. We introduced new scoring functions for the judge network that makes the training more stable. Our experimental results demonstrate that: (1) DAN can boost the performance when using a small number of labeled samples; (2) the loss functions learned by DAN outperform standard-pairwise and negative log-likelihood loss functions for the semi-supervised setup, and is also very competitive in the supervised setting.

2 Methods

2.1 Generative Adversarial Nets and Conditional GANs

Generative adversarial networks are an effective approach for training generative models [7]. The GAN framework normally comprises two "adversarial" networks: a generative net G that "learns" the data distribution, and a discriminative net D that estimates the probability that a sample came from the real data distribution rather than generated by G. In order to learn a generator distribution p_g over data x, the generator builds a mapping function from a prior noise distribution $p_z(z)$ to the data space as $G(z;\theta_g)$. The discriminator receives as input a data point x and outputs a single scalar, $D(x;\theta_d)$, which represents the probability that x came from training data rather than p_g . G and D are trained simultaneously by adjusting parameters for G to minimize $log(1-D(G(z))$ and adjusting parameters for D to minimize $logD(x)$, as if they are following a two-player min-max game with the following value function $V(G,D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[logD(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[log(1-D(G(z)))\right] \quad (1)$$

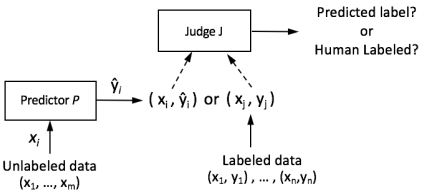
Generative adversarial nets can perform conditional generation if both the generator and discriminator are conditioned on some extra information y [19], generally a class label. Normally y is a class label and the conditioning is performed by feeding y into both the discriminator and generator as an additional input. In the generator, the prior input noise $p_z(z)$ and y are combined in a joint hidden representation. Usually this consists of simply concatenating a vector representation of y to the input vector z. The discriminator receives both x and y as inputs and has to discriminate between real x, y and generated $G(z,y)$. The objective function of the two-player minimax game can be formulated as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x,y\sim p_{data}(x,y)}\left[logD(x,y)\right] + \mathbb{E}_{z\sim p_z(z),y\sim p_y(y)}\left[log(1-D(G(z,y),y))\right] \quad (2)$$

2.2 Discriminative Adversarial Networks

We call DAN the adversarial network framework that uses discriminators only. In our DAN formulation (Fig. 1) we use two discriminators: the *Predictor* P and the *Judge* J. P receives as input a data point x and outputs a prediction $P(x)$. The prediction can be a simple probability distribution over class labels or any other sort of structured predictions such as trees or document rankings. The Judge network J receives as input a data point x and a label y [2] and produces a single scalar, $J(x,y)$, which represents the

Figure 1: DAN framework.



[2]We are using the term *label* in a loose way to also mean any type of structured prediction.

probability that x, y came from the labeled training data
rather than predicted by P. While in conditional GANs
the idea is to generate x conditioned on y, in DAN we want to predict y conditioned on x. The min-max game value function V(J,P) becomes:

$$\min_P \max_J V(J,P) = \mathbb{E}_{x,y\sim p_{data}(x,y)}\ [logJ(x,y)] + \mathbb{E}_{x\sim p_{data}(x)}\ [log(1-J(x,P(x)))] \quad (3)$$

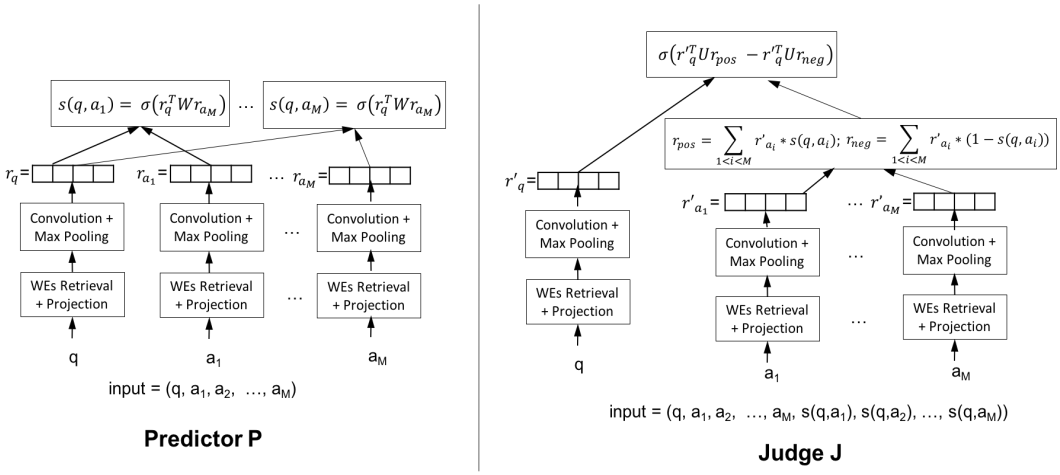


Figure 2: DAN Architecture for Answer Selection

2.2.1 DAN for Answer Selection / Ranking

In the answer selection task, given a question q and a candidate answer pool $P = (a_1, a_2, \cdots, a_M)$ for q, the goal is to search for and select the candidate answer(s) $a \in P$ that correctly answers q. This task can be viewed as a ranking problem where the goal is to rank the candidate answers from the best to the worst. People normally use the following pairwise ranking loss function (hinge loss) when optimizing neural network based rankers: $L = \max\{0, l - s_\theta(q,a^+) + s_\theta(q,a^-)\}$, where a^+ is a positive answer, a^- is a negative answer and l is a margin. However, pairwise ranking loss is known to be suboptimal [2]. Our goal on choosing this ranking task is two fold: (1) we believe that the semi-supervised nature of DANs can help to reduce the need of labeled data for answer selection; (2) we believe that DANs can learn a good listwise loss function by taking into consideration the scoring of the whole set of candidate answers.

As depicted in the left hand side of Fig. 2, the Predictor P takes as input the list $(q, a_1, a_2, \ldots, a_M)$ containing a question q and M candidate answers. Pairs of (q, a_i) are processed in parallel by first generating fixed-length continuous vector representations r_q and r_{a_i} and then performing the operation $\sigma(r_q^T W r_{a_i})$, where W is a matrix of learnable parameters and σ is the sigmoid function. Since we are using a sigmoid, note that the score produced by P is a number between 0 and 1. The parameters of WE projection layer, convolution layer and W are shared among question and all candidate answers.

The right hand side of Fig. 2 details the Judge J, which uses a similar architecture as the predictor, except for the scoring function. There is no parameter sharing between P and J. Note that J also receives as input the score for each candidate answer, which means that J performs a listwise evaluation. For labeled instances, the score for a correct answer is 1 and for an incorrect answer is 0.

After creating the representation r'_q, r'_{a_1}, ..., r'_{a_m}, the Judge J uses the scores s_{a_1}, ..., s_{a_m} to compute representations r_{pos} and r_{neg} as follows:

$$r_{pos} = \sum_{1<i<=M} r'_{a_i} * s(q,a_i) \qquad r_{neg} = \sum_{1<i<=M} r'_{a_i} * (1 - s(q,a_i)) \quad (4)$$

We can think of r_{pos} and r_{neg} as a way to summarize, according to the scores, the similarities and dissimilarities, respectively, between the question and the list of candidate answers. The final scoring is given by $\sigma(r'^T_q U r_{pos} - r'^T_q U r_{neg})$. The rationale behind this scoring function is that, if the given

list of scores is good, the representation of the question, r'_q, should be more similar to r_{pos} than to r_{neg}. As far as we know, this scoring function is novel, and we further extended it for the classification task as presented in the next section.

2.2.2 DAN for Text Classification

Our DAN architecture for text classification is similar to the answer selection architecture in many aspects. We use one layer CNNs on P and J, and J employs a scoring function that is similar to the one used in the answer selection architecture. More architecture details are in the Appendix A.1.

3 Related work

Our approach is mainly related to recent works on semi-supervised GANs and conditional GANs. Springenberg [26] proposed a categorical generative adversarial network (CatGAN) which can be used for unsupervised and semi-supervised learning, where the discriminator outputs a distribution over classes and is trained to minimize the predicted entropy for real data and maximize the predicted entropy for fake data. Salimans et al. [24] proposed a semi-supervised GAN model in which the discriminator outputs a softmax over classes rather than a probability of real vs. fake. An additional "generated" class is used as the target for generated samples. Kumar et al. [15] use the same GAN-like SSL setup proposed in [24], but use tangents from the generator's mapping to further improve on SSL. Different from these works, in DAN we do not perform a generation step, therefore it is easier to apply for discrete data.

Regarding loss function learning using GAN-like approaches, Isola et al. [9] proposed conditional GANs for image-to-image translation problems, and showed that their models not only learns good mappings but also learns a loss function to train the mapping. Finn et al. [6] presented a connection between GAN-based loss function learning for generative models and cost function learning in reinforcement learning (aka inverse reinforcement learning). They demonstrated that certain IRL methods are mathematically equivalent to GANs. While previous work focus on learning loss functions for generative models, in DAN we focus on learning loss functions for discriminative models.

Another recent line of work consists of using adversarial examples [27, 8] based on unlabeled data to regularize the training [20]. For the NLP domain, the work by Miyato et al. [21] extended the adversarial and virtual adversarial training approaches by adding small perturbations to embeddings. They report good performance for semi-supervised text classification tasks. In DAN, instead of adding an extra regularization term to the supervised loss, we implicitly learn the loss function.

4 Experiments and results

We use two different datasets to perform our answer selection experiments: SelQA [10] and WikiQA [29]. For both datasets, we use the subtask that assumes that there is at least one correct answer for a question. For the text classification task, we use the Stanford Sentiment Tree-bank (SSTb) dataset and focus on binary classification only.

For both tasks, answer selection and sentiment classification, we perform semi-supervised experiments where we randomly sample a limited number of labeled instances and use the rest as unlabeled data. We use the term CNN-DAN to refer to the DAN architecture for that respective task. However, in the CNN-DAN setup, the instances presented to P are the exact same instances that appear in the labeled set. Therefore, CNN-DAN is basically trying to learn a better loss function using the available labeled data, no semi-supervised learning is performed. We use the term CNN-DAN$_{unlab.}$ to refer to the DAN setup where we feed P with additional unlabeled data.

In Tables 1, 2 and 3 we present the experimental results for SelQA, WikiQA and SSTB2, respectively. We present results for: CNN-DAN; CNN-DAN$_{unlab.}$, that uses unlabeled data in P; CNN$_{hinge_loss}$, which is the same CNN-based architecture of the predictor P in our DAN for answer selection (Fig. 2), but that is trained using the hinge loss function instead of the DAN framework; CNN$_{nll}$, which is the same CNN-based architecture of the predictor P in our DAN for text classification (Fig. 6), but that is trained using the negative log likelihood (NLL) loss function instead of the DAN framework. We present detailed results for datasets containing a different number of labeled instances: 10, 50 and

Table 1: Experimental Results for the SelQA dataset.

Model	10 labeled instances			50 labeled instances			Full dataset		
	MAP	MRR	NDCG	MAP	MRR	NDCG	MAP	MRR	NDCG
CNN_{hinge_loss}	.4610	.4661	.5889	.6455	.6545	.7331	**.8758**	**.8812**	**.9079**
CNN-DAN	.5749	.5811	.6780	.6248	.6332	.7170	.8655	.8730	.9012
$\text{CNN-DAN}_{unlab.}$	**.6891**	**.6978**	**.7667**	**.6928**	**.7017**	**.7695**	-	-	-

Table 2: Experimental Results for the WikiQA dataset.

Model	10 labeled instances			50 labeled instances			Full dataset		
	MAP	MRR	NDCG	MAP	MRR	NDCG	MAP	MRR	NDCG
CNN_{hinge_loss}	.5447	.5577	.6575	.5919	.6071	.6942	.6511	.6669	.7402
CNN-DAN	.5437	.5582	.6566	.6047	.6201	.7042	**.6663**	**.6822**	**.7516**
$\text{CNN-DAN}_{unlab.}$	**.5927**	**.6068**	**.6945**	**.6127**	**.6274**	**.7104**	-	-	-

Table 3: Experimental Results for the SSTB2 dataset.

Model	10 instances	50 instances	Full dataset
	Average Accuracy	Average Accuracy	Average Accuracy
CNN_{nll}	$60.42 \pm (3.68)$	$71.36 \pm (3.04)$	**87.0**
CNN-DAN	$63.25 \pm (3.78)$	$\mathbf{74.23 \pm (2.02)}$	**87.1**
$\text{CNN-DAN}_{unlab.}$	$\mathbf{65.62 \pm (3.46)}$	$72.37 \pm (1.79)$	-

Table 4: Comparison with previously reported results for SelQA, WikiQA and SSTB2.

SelQA			WikiQA			SSTB2	
Model	MAP	MRR	Model	MAP	MRR	Model	Acc.
[10]	.8643	**.8759**	[29]	.6520	.6652	[3]	85.5
CNN-DAN	**.8655**	.8730	[31]	.6600	.6770	CNN-DAN	**87.1**
-	-	-	CNN-DAN	.6663	.6822	[11]	**87.2**
-	-	-	[4]	**.6886**	**.6957**		

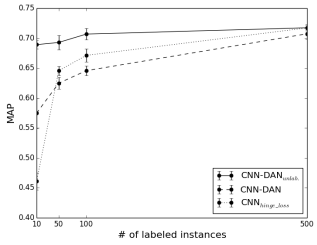


Figure 3: SelQA

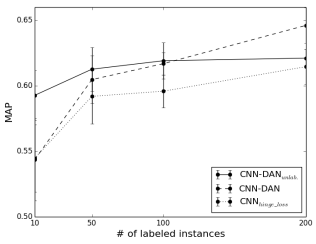


Figure 4: WikiQA

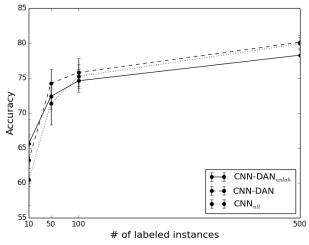


Figure 5: SSTB2

full dataset. In Figs. 3 and 4 we also present the MAP, and the Acc. in Fig 5, for datasets with 100 and 500 labeled instances.

We can see in Figs. 3, 4 and 5 that the semi-supervised DAN, $\text{CNN-DAN}_{unlab.}$, gives a significant boost in performance when a small amount of labeled instances is available. These results are evidence that DAN is a promising approach for semi-supervised learning. Comparing CNN-DAN, that does not use unlabeled data, with CNN_{hinge_loss} and CNN_{NLL} is a reasonable way to check whether the learned loss function is doing better or not than the pairwise hinge loss and the NLL loss. We can see that in most situations the loss function implicitly learned by the judge performs at least as good as the NLL loss and the pairwise hinge loss.

References

[1] A. Bordes, J. Weston, and N. Usunier. Open question answering with weakly supervised embedding models. In *Proc. of ECML*, pages 165–180, 2014.

[2] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *Proc. of ICML*, ICML '07, pages 129–136, 2007.

[3] C. N. dos Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proc. of COLING*, pages 69–78, 2014.

[4] C. N. dos Santos, M. Tan, B. Xiang, and B. Zhou. Attentive pooling networks. *CoRR, abs/1602.03609*, 2016.

[5] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville. Adversarially learned inference. In *Proc. of ICLR*, 2017.

[6] C. Finn, P. Christiano, P. Abbeel, and S. Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. In *NIPS Workshop on Adversarial Training*, 2016.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proc. of NIPS*, page 2672, 2014.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[9] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proc. of CVPR*, 2017.

[10] T. Jurczyk, M. Zhai, and J. D. Choi. Selqa: A new benchmark for selection-based question answering. *arXiv preprint arXiv:1606.08513*, 2016.

[11] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[13] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Proc. of NIPS*, pages 3581–3589, 2014.

[15] A. Kumar, P. Sattigeri, and P. T. Fletcher. Improved semi-supervised learning with gans using manifold invariances. *CoRR*, abs/1705.08850, 2017.

[16] L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. Auxiliary deep generative models. In *Proc. of ICML*, pages 1445–1453, 2016.

[17] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proc. of ACL*, pages 142–150, 2011.

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

[19] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[20] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*, 2015.

[21] T. Miyato, A. M. Dai, and I. Goodfellow. Virtual adversarial training for semi-supervised text classification. In *Proc. of ICLR*, 2016.

[22] A. Odena. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

[23] M. Sajjadi, M. Javanmardi, and T. Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Proc. of NIPS*, pages 1163–1171, 2016.

[24] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Proc. of NIPS*, pages 2226–2234, 2016.

[25] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*, page 1642, 2013.

[26] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *Proc. of ICLR*, 2016.

[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[28] W. Xu, H. Sun, C. Deng, and Y. Tan. Variational autoencoder for semi-supervised text classification. In *Proc. of AAAI*, 2017.

[29] Y. Yang, W.-t. Yih, and C. Meek. Wikiqa: A challenge dataset for open-domain question answering. In *Proc. of EMNLP*, pages 2013–2018, 2015.

[30] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. In *Proc. of ICML*, 2017.

[31] W. Yin, H. Schütze, B. Xiang, and B. Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *arXiv preprint arXiv:1512.05193*, 2015.

[32] X. Zhu. Semi-supervised learning literature survey. Technical report, Computer Sciences, University of Wisconsin-Madison, 2005.

A Appendix

A.1 DAN Architecture for Text Classification

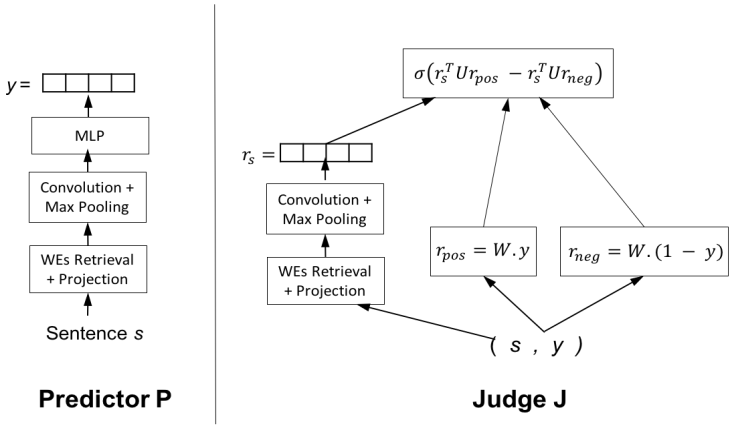


Figure 6: DAN Architecture for Text Classification

As illustrated in left hand side of Fig. 6, the Predictor P is a standard CNN-based text classifier that classifies a given sentence s into one of N classes. It takes in sentence s as input and outputs y a probability distribution over N classes. We first retrieve the word embeddings (WEs) and project them using a fully connected layer. Next, a convolutional layer followed by a MLP is used to perform the prediction.

The Judge J takes in a pair (x,y) consisting of a sentence and its class label, and classifies the pair as being predicted label (fake) or human labeled pair (real). For the human labeled pairs, y is encoded as the one hot representation of the class label. For predicted pairs, $\hat{y}$ is the output of P, which is a probability distributions over the class labels. As in the Predictor, we create a representation r_s of the sentence using a convolution. In the Judge, as can be noticed in right hand side of Fig. 6, we create two representations of the class label y, r_{pos} and r_{neg} using a embedding matrix W. The representation r_{pos}, can be seen as the embedding of the positive/correct label. While the representation r_{neg} can be understood as the average embedding of the negative classes. The final scoring is done by first measuring the similarity between r_s and r_{pos}, and between r_s and r_{neg} using bilinear forms: $r_s^T U r_{pos}$ and $r_s^T U r_{pos}$, where U is a matrix of learnable parameters. This type of bilinear similarity measure has been previously used for embedding comparison in [1]. Next, the difference between the two similarities are passed through the sigmoid function (σ). The rationale behind this scoring function is that, if the given label is correct, the representation of the sentence, r_s, should be more similar to r_{pos} than to r_{neg}. In our experiments, this scoring approach has shown to be empirically easier to train under the min-max game than concatenating r_{pos} and r_s and giving the resulting vector as input to a logistic regression (or MLP). We developed this scoring approach for the ranking task first (next section) and later we realized that it also works well for classification.

A.2 Experimental setup details

For both SelQA and WikiQA datasets, we use the subtask that assumes that there is at least one correct answer for a question. For the WikiQA, the corresponding dataset consists of 873 questions in the training set (20,360 question/candidate pairs), 126 in the dev set (1,130 pairs) and 243 questions in the test set (2,352 pairs). For SelQA, the corresponding dataset contains of 5529 questions in the training set, 785 in the dev set and 1590 questions in the test set. SelQA is more than 6 times larger than WikiQA in number of questions.

The Stanford Sentiment Tree-bank (SSTb) dataset is a movie review dataset proposed by [25] which includes fine grained sentiment labels for 215,154 phrases in the parse trees of 11,855 sentences. In our experiments we focus on binary classification only. This dataset, which is known as SSTB2, contains 6,920 training sentences, 872 dev. sentences and 1,821 test sentences. When performing experiments with the full dataset we follow the protocol used in [25], which consists in traininig with the set of all phrases instead of just the complete sentences. However, in the semi-supervised experiments we train using complete sentences only.

In all experiments, we use word embeddings of size 400, which were pre-trained using the word2vec tool [18]. For the answer selection task we use a dump of Enlgish Wikipedia. For sentiment classification, we pretrain the word embeddings using the IMDB data proposed by [17].

We use the ADAM optimizer [12], and kept the values of most of the hyperparameters fixed for all the experiments. For both the Predictor and the Judge, the word embeddings projection layer has 200 units, the convolutional layer has 400 filters, with context window of sizes 3 and 5 words in the case of answer selection and text classification, respectively. The U matrix has dimensionally $\mathbb{R}^{400\times400}$. When training using the full dataset, we alternately update J and P one time each. We use a learning rate of $\lambda = 0.0005$ for the answer selection task, and $\lambda = 0.0001$ for the text classification task. Validation sets are used to perform early stopping. Normally it is needed less than 100 epochs to achieve the best performance in the validation set.

For the semi-supervised experiments, since the set of unlabeled instances is much large than the one of labeled, we noticed that we need to update P more frequently than J in order to avoid overfitting J. For better results in the semi-supervised setup, we normally update P 10 times after each update of J. However, in this case we also had to use a smaller learning rates for P ($\lambda = 0.00005$) and J ($\lambda = 0.0001$).

For both tasks, answer selection and sentiment classification, we perform semi-supervised experiments where we randomly sample a limited number of labeled instances and use the rest of the dataset as unlabeled data. In all experiments reported in the next two sections, we repeat the random sampling 10 times and average the results. Additionally, in the experiments using the full labeled dataset we repeat the experiments 10 times with different seeds for the random number generator and average the results.

For both tasks, we use the term CNN-DAN to refer to the DAN architecture for that respective task (Figs. 6 and 2). However, in the CNN-DAN setup, the instances presented to P are the exact same instances that appear in the labeled set. Therefore, CNN-DAN is basically trying to learn a better loss function using the available labeled data, no semi-supervised learning is performed. We use the term CNN-DAN$_{unlab.}$ to refer to the DAN setup where we feed P with additional unlabeled data.