Domain Adaptation with Adversarial Training and Graph Embeddings

Firoj Alam[*], **Shafiq Joty**[†], and **Muhammad Imran**[*]
Qatar Computing Research Institute, HBKU, Qatar[*]
School of Computer Science and Engineering†
Nanyang Technological University, Singapore[†]
{fialam, mimran}@hbku.edu.qa[*]
srjoty@ntu.edu.sg[†]

Abstract

The success of deep neural networks (DNNs) is heavily dependent on the availability of labeled data. However, obtaining labeled data is a big challenge in many real-world problems. In such scenarios, a DNN model can leverage labeled and unlabeled data from a related domain, but it has to deal with the shift in data distributions between the source and the target domains. In this paper, we study the problem of classifying social media posts during a crisis event (e.g., Earthquake). For that, we use labeled and unlabeled data from past similar events (e.g., Flood) and unlabeled data for the current event. We propose a novel model that performs adversarial learning based domain adaptation to deal with distribution drifts and graph based semi-supervised learning to leverage unlabeled data within a single unified deep learning framework. Our experiments with two real-world crisis datasets collected from Twitter demonstrate significant improvements over several baselines.

1 Introduction

The application that motivates our work is the time-critical analysis of social media (Twitter) data at the sudden-onset of an event like natural or man-made disasters (Imran et al., 2015). In such events, affected people post timely and useful information of various types such as reports of injured or dead people, infrastructure damage, urgent needs (e.g., food, shelter, medical assistance) on these social networks. Humanitarian organizations believe timely access to this important information from social networks can help significantly and reduce both human loss and economic damage (Varga et al., 2013; Vieweg et al., 2014; Power et al., 2013).

In this paper, we consider the basic task of classifying each incoming tweet during a crisis event (e.g., Earthquake) into one of the predefined classes of interest (e.g., *relevant* vs. *non-relevant*) in real-time. Recently, deep neural networks (DNNs) have shown great performance in classification tasks in NLP and data mining. However the success of DNNs on a task depends heavily on the availability of a large labeled dataset, which is not a feasible option in our setting (i.e., classifying tweets at the onset of an Earthquake). On the other hand, in most cases, we can have access to a good amount of labeled and abundant unlabeled data from past similar events (e.g., Floods) and possibly some unlabeled data for the current event. In such situations, we need methods that can leverage the labeled and unlabeled data in a past event (we refer to this as a *source* domain), and that can adapt to a new event (we refer to this as a *target* domain) without requiring any labeled data in the new event. In other words, we need models that can do *domain adaptation* to deal with the distribution drift between the domains and *semi-supervised* learning to leverage the unlabeled data in both domains.

Most recent approaches to semi-supervised learning (Yang et al., 2016) and domain adaptation (Ganin et al., 2016) use the automatic feature learning capability of DNN models. In this paper, we extend these methods by proposing a novel model that performs domain adaptation and semi-supervised learning within a single unified deep learning framework. In this framework, the basic task-solving network (a convolutional neural network in our case) is put together with two other networks – one for semi-supervised learning and the other for domain adaptation. The semi-supervised component learns internal representa-

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1077–1087
Melbourne, Australia, July 15 - 20, 2018.

tions (features) by predicting contextual nodes in a graph that encodes *similarity* between labeled and unlabeled training instances. The domain adaptation is achieved by training the feature extractor (or encoder) in *adversary* with respect to a domain discriminator, a binary classifier that tries to distinguish the domains. The overall idea is to learn high-level abstract representation that is discriminative for the main classification task, but is invariant across the domains. We propose a stochastic gradient descent (SGD) algorithm to train the components of our model simultaneously.

The evaluation of our proposed model is conducted using two Twitter datasets on scenarios where there is only unlabeled data in the target domain. Our results demonstrate the following.

1. When the network combines the semi-supervised component with the supervised component, depending on the amount of labeled data used, it gives 5% to 26% absolute gains in F1 compared to when it uses only the supervised component.
2. Domain adaptation with adversarial training improves over the adaptation baseline (i.e., a transfer model) by 1.8% to 4.1% absolute F1.
3. When the network combines domain adversarial training with semi-supervised learning, we get further gains ranging from 5% to 7% absolute in F1 across events.

Our source code is available on Github[1] and the data is available on CrisisNLP[2].

The rest of the paper is organized as follows. In Section 2, we present the proposed method, i.e., domain adaptation and semi-supervised graph embedding learning. In Section 3, we present the experimental setup and baselines. The results and analysis are presented in Section 4. In Section 5, we present the works relevant to this study. Finally, conclusions appear in Section 6.

2 The Model

We demonstrate our approach for domain adaptation with adversarial training and graph embedding on a tweet classification task to support crisis response efforts. Let $\mathcal{D}_S^l = \{\mathbf{t}_i, y_i\}_{i=1}^{L_s}$ and $\mathcal{D}_S^u = \{\mathbf{t}_i\}_{i=1}^{U_s}$ be the set of labeled and unlabeled tweets for a source crisis event S (e.g., Nepal earthquake), where $y_i \in \{1, \ldots, K\}$ is the class label for tweet $\mathbf{t}_i$, L_s and U_s are the number of labeled and unlabeled tweets for the source event, respectively. In addition, we have unlabeled tweets $\mathcal{D}_T^u = \{\mathbf{t}_i\}_{i=1}^{U_t}$ for a target event T (e.g., Queensland flood) with U_t being the number of unlabeled tweets in the target domain. Our ultimate goal is to train a cross-domain model $p(y|\mathbf{t}, \theta)$ with parameters θ that can classify any tweet in the target event T without having any information about class labels in T.

Figure 1 shows the overall architecture of our neural model. The input to the network is a tweet $\mathbf{t} = (w_1, \ldots, w_n)$ containing words that come from a finite vocabulary $\mathcal{V}$ defined from the training set. The first layer of the network maps each of these words into a distributed representation $\mathbb{R}^d$ by looking up a shared embedding matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d}$. We initialize the embedding matrix E in our network with word embeddings that are pre-trained on a large crisis dataset (Subsection 2.5). However, embedding matrix E can also be initialize randomly. The output of the look-up layer is a matrix $X \in \mathbb{R}^{n \times d}$, which is passed through a number of convolution and pooling layers to learn higher-level feature representations. A *convolution* operation applies a *filter* $\mathbf{u} \in \mathbb{R}^{k.d}$ to a window of k vectors to produce a new feature h_t as

$$h_t = f(\mathbf{u}.X_{t:t+k-1}) \tag{1}$$

where $X_{t:t+k-1}$ is the concatenation of k look-up vectors, and f is a nonlinear activation; we use rectified linear units or ReLU. We apply this filter to each possible k-length windows in X with stride size of 1 to generate a *feature map* $\mathbf{h}^j$ as:

$$\mathbf{h}^j = [h_1, \ldots, h_{n+k-1}] \tag{2}$$

We repeat this process N times with N different filters to get N different feature maps. We use a *wide* convolution (Kalchbrenner et al., 2014), which ensures that the filters reach the entire tweet, including the boundary words. This is done by performing *zero-padding*, where out-of-range (i.e., $t<1$ or $t>n$) vectors are assumed to be zero. With wide convolution, o zero-padding size and 1 stride size, each feature map contains $(n + 2o - k + 1)$ convoluted features. After the convolution, we apply a *max-pooling* operation to each of the feature maps,

$$\mathbf{m} = [\mu_p(\mathbf{h}^1), \cdots, \mu_p(\mathbf{h}^N)] \tag{3}$$

[1] https://github.com/firojalam/domain-adaptation

[2] http://crisisnlp.qcri.org

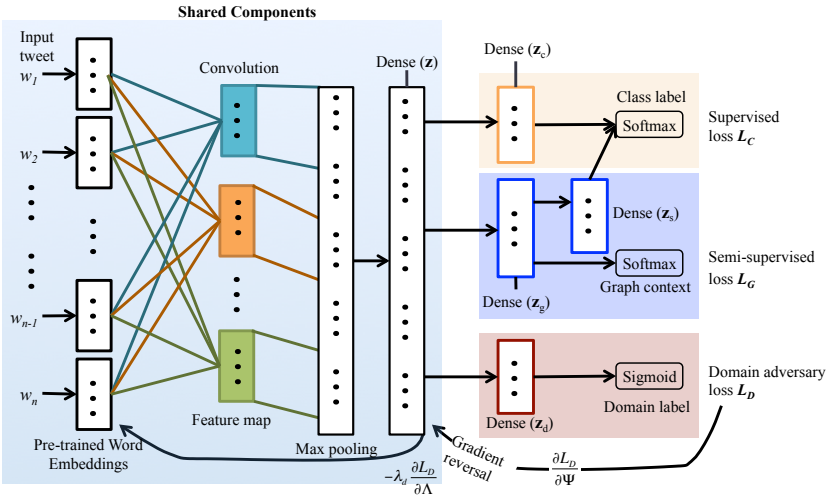

Figure 1: The system architecture of the domain adversarial network with graph-based semi-supervised learning. The shared components part is shared by supervised, semi-supervised and domain classifier.

where $\mu_p(\mathbf{h}^j)$ refers to the max operation applied to each window of p features with stride size of 1 in the feature map $\mathbf{h}^i$. Intuitively, the convolution operation composes local features into higher-level representations in the feature maps, and max-pooling extracts the most important aspects of each feature map while reducing the output dimensionality. Since each convolution-pooling operation is performed independently, the features extracted become invariant in order (i.e., where they occur in the tweet). To incorporate order information between the pooled features, we include a fully-connected (dense) layer

$$\mathbf{z} = f(V\mathbf{m}) \tag{4}$$

where V is the weight matrix. We choose a convolutional architecture for feature composition because it has shown impressive results on similar tasks in a supervised setting (Nguyen et al., 2017).

The network at this point splits into three branches (shaded with three different colors in Figure 1) each of which serves a different purpose and contributes a separate loss to the overall loss of the model as defined below:

$$\mathcal{L}(\Lambda, \Phi, \Omega, \Psi) = \mathcal{L}_C(\Lambda, \Phi) + \lambda_g \mathcal{L}_G(\Lambda, \Omega) + \lambda_d \mathcal{L}_D(\Lambda, \Psi) \tag{5}$$

where $\Lambda = \{U, V\}$ are the convolutional filters and dense layer weights that are shared across the three branches. The first component $\mathcal{L}_C(\Lambda, \Phi)$ is a supervised classification loss based on the labeled data in the source event. The second component $\mathcal{L}_G(\Lambda, \Omega)$ is a graph-based semi-supervised loss that utilizes both labeled and unlabeled data in the source and target events to induce structural similarity between training instances. The third component $\mathcal{L}_D(\Lambda, \Omega)$ is an adversary loss that again uses all available data in the source and target domains to induce domain invariance in the learned features. The tunable hyperparameters λ_g and λ_d control the relative strength of the components.

2.1 Supervised Component

The supervised component induces label information (e.g., *relevant* vs. *non-relevant*) directly in the network through the classification loss $\mathcal{L}_C(\Lambda, \Phi)$, which is computed on the labeled instances in the source event, $\mathcal{D}_S^l$. Specifically, this branch of the network, as shown at the top in Figure 1, takes the shared representations $\mathbf{z}$ as input and pass it through a task-specific dense layer

$$\mathbf{z}_c = f(V_c\mathbf{z}) \tag{6}$$

where V_c is the corresponding weight matrix. The activations $\mathbf{z}_c$ along with the activations from the semi-supervised branch $\mathbf{z}_s$ are used for classification. More formally, the classification layer defines a Softmax

$$p(y = k|\mathbf{t}, \theta) = \frac{\exp\left(W_k^T [\mathbf{z}_c; \mathbf{z}_s]\right)}{\sum_{k'} \exp\left(W_{k'}^T [\mathbf{z}_c; \mathbf{z}_s]\right)} \tag{7}$$

where $[.;.]$ denotes concatenation of two column vectors, W_k are the class weights, and $\theta = \{U, V, V_c, W\}$ defines the relevant parameters for this branch of the network with $\Lambda = \{U, V\}$ being the shared parameters and $\Phi = \{V_c, W\}$ being the parameters specific to this branch. Once learned,

we use θ for prediction on test tweets. The classification loss $\mathcal{L}_C(\Lambda, \Phi)$ (or $\mathcal{L}_C(\theta)$) is defined as

$$\mathcal{L}_C(\Lambda, \Phi) = -\frac{1}{L_s} \sum_{i=1}^{L_s} \mathbb{I}(y_i = k) \log p(y_i = k | \mathbf{t}_i, \Lambda, \Phi) \quad (8)$$

where $\mathbb{I}(.)$ is an indicator function that returns 1 when the argument is true, otherwise it returns 0.

2.2 Semi-supervised Component

The semi-supervised branch (shown at the middle in Figure 1) induces structural similarity between training instances (labeled or unlabeled) in the source and target events. We adopt the recently proposed graph-based semi-supervised deep learning framework (Yang et al., 2016), which shows impressive gains over existing semi-supervised methods on multiple datasets. In this framework, a "similarity" graph G first encodes relations between training instances, which is then used by the network to learn internal representations (i.e., embeddings).

2.2.1 Learning Graph Embeddings

The semi-supervised branch takes the shared representation $\mathbf{z}$ as input and learns internal representations by predicting a node in the graph context of the input tweet. Following (Yang et al., 2016), we use *negative sampling* to compute the loss for predicting the context node, and we sample two types of contextual nodes: (*i*) one is based on the graph G to encode structural information, and (*ii*) the second is based on the labels in $\mathcal{D}_S^l$ to incorporate label information through this branch of the network. The ratio of positive and negative samples is controlled by a random variable $\rho_1 \in (0, 1)$, and the proportion of the two context types is controlled by another random variable $\rho_2 \in (0, 1)$; see Algorithm 1 of (Yang et al., 2016) for details on the sampling procedure.

Let (j, γ) is a tuple sampled from the distribution $p(j, \gamma | i, \mathcal{D}_S^l, G)$, where j is a context node of an input node i and $\gamma \in \{+1, -1\}$ denotes whether it is a positive or a negative sample; $\gamma = +1$ if $\mathbf{t}_i$ and $\mathbf{t}_j$ are neighbors in the graph (for graph-based context) or they both have same labels (for label-based context), otherwise $\gamma = -1$. The negative log loss for context prediction $\mathcal{L}_G(\Lambda, \Omega)$ can be written as

$$\mathcal{L}_G(\Lambda, \Omega) = -\frac{1}{L_s + U_s} \sum_{i=1}^{L_s+U_s} \mathbb{E}_{(j,\gamma)} \log \sigma \left(\gamma C_j^T \mathbf{z}_g(i) \right) \quad (9)$$

where $\mathbf{z}_g(i) = f(V_g \mathbf{z}(i))$ defines another dense layer (marked as *Dense ($\mathbf{z}_g$)* in Figure 1) having weights V_g, and C_j is the weight vector associated with the context node $\mathbf{t}_j$. Note that here $\Lambda = \{U, V\}$ defines the shared parameters and $\Omega = \{V_g, C\}$ defines the parameters specific to the semi-supervised branch of the network.

2.2.2 Graph Construction

Typically graphs are constructed based on a relational knowledge source, e.g., citation links in (Lu and Getoor, 2003), or distance between instances (Zhu, 2005). However, we do not have access to such a relational knowledge in our setting. On the other hand, computing distance between $n(n-1)/2$ pairs of instances to construct the graph is also very expensive (Muja and Lowe, 2014). Therefore, we choose to use k-nearest neighbor-based approach as it has been successfully used in other study (Steinbach et al., 2000).

The nearest neighbor graph consists of n vertices and for each vertex, there is an edge set consisting of a subset of n instances, i.e., tweets in our training set. The edge is defined by the distance measure $d(i, j)$ between tweets $\mathbf{t}_i$ and $\mathbf{t}_j$, where the value of d represents how similar the two tweets are. We used k-d tree data structure (Bentley, 1975) to efficiently find the nearest instances. To construct the graph, we first represent each tweet by averaging the word2vec vectors of its words, and then we measure $d(i, j)$ by computing the *Euclidean* distance between the vectors. The number of nearest neighbor k was set to 10. The reason of averaging the word vectors is that it is computationally simpler and it captures the relevant semantic information for our task in hand. Likewise, we choose to use Euclidean distance instead of cosine for computational efficiency.

2.3 Domain Adversarial Component

The network described so far can learn abstract features through convolutional and dense layers that are discriminative for the classification task (*relevant* vs. *non-relevant*). The supervised branch of the network uses labels in the source event to induce label information directly, whereas the semi-supervised branch induces similarity information between labeled and unlabeled instances. However, our goal is also to make these learned features invariant across domains or events (e.g., *Nepal Earthquake* vs. *Queensland Flood*). We achieve this by domain *adversarial* training of

neural networks (Ganin et al., 2016).

We put a domain discriminator, another branch in the network (shown at the bottom in Figure 1) that takes the shared internal representation $\mathbf{z}$ as input, and tries to discriminate between the domains of the input — in our case, whether the input tweet is from $\mathcal{D}_S$ or from $\mathcal{D}_T$. The domain discriminator is defined by a sigmoid function:

$$\hat{\delta} = p(d = 1|\mathbf{t}, \Lambda, \Psi) = \text{sigm}(\mathbf{w}_d^T \mathbf{z}_d) \quad (10)$$

where $d \in \{0, 1\}$ denotes the domain of the input tweet $\mathbf{t}$, $\mathbf{w}_d$ are the final layer weights of the discriminator, and $\mathbf{z}_d = f(V_d\mathbf{z})$ defines the hidden layer of the discriminator with layer weights V_d. Here $\Lambda = \{U, V\}$ defines the shared parameters, and $\Psi = \{V_d, \mathbf{w}_d\}$ defines the parameters specific to the domain discriminator. We use the negative log-probability as the discrimination loss:

$$\mathcal{J}_i(\Lambda, \Psi) = -d_i \log \hat{\delta} - (1 - d_i) \log \left(1 - \hat{\delta}\right) \quad (11)$$

We can write the overall domain adversary loss over the source and target domains as

$$\mathcal{L}_D(\Lambda, \Psi) = -\frac{1}{L_s + U_s} \sum_{i=1}^{L_s+U_s} \mathcal{J}_i(\Lambda, \Psi) - \frac{1}{U_t} \sum_{i=1}^{U_t} \mathcal{J}_i(\Lambda, \Psi) \quad (12)$$

where $L_s + U_s$ and U_t are the number of training instances in the source and target domains, respectively. In adversarial training, we seek parameters (saddle point) such that

$$\theta^* = \underset{\Lambda,\Phi,\Omega}{\text{argmin}} \max_{\Psi} \mathcal{L}(\Lambda, \Phi, \Omega, \Psi) \quad (13)$$

which involves a maximization with respect to Ψ and a minimization with respect to $\{\Lambda, \Phi, \Omega\}$. In other words, the updates of the shared parameters $\Lambda = \{U, V\}$ for the discriminator work adversarially to the rest of the network, and vice versa. This is achieved by reversing the gradients of the discrimination loss $\mathcal{L}_D(\Lambda, \Psi)$, when they are backpropagated to the shared layers (see Figure 1).

2.4 Model Training

Algorithm 1 illustrates the training algorithm based on stochastic gradient descent (SGD). We first initialize the model parameters. The word embedding matrix E is initialized with pre-trained word2vec vectors (see Subsection 2.5) and is kept fixed during training.[3] Other parameters are initialized with small random numbers sampled from a uniform distribution (Bengio and Glorot, 2010). We use AdaDelta (Zeiler, 2012) adaptive update to update the parameters.

Algorithm 1: Model Training with SGD

Input : data $\mathcal{D}_S^l$, $\mathcal{D}_S^u$, $\mathcal{D}_T^u$; graph G
Output: learned parameters $\theta = \{\Lambda, \Phi\}$
1. Initialize model parameters $\{E, \Lambda, \Phi, \Omega, \Psi\}$;
2. **repeat**
  `// Semi-supervised`
  for *each batch sampled from* $p(j, \gamma|i, \mathcal{D}_S^l, G)$ **do**
    a) Compute loss $\mathcal{L}_G(\Lambda, \Omega)$
    b) Take a gradient step for $\mathcal{L}_G(\Lambda, \Omega)$;
  end
  `// Supervised & domain adversary`
  for *each batch sampled from* $\mathcal{D}_S^l$ **do**
    a) Compute $\mathcal{L}_C(\Lambda, \Phi)$ and $\mathcal{L}_D(\Lambda, \Psi)$
    b) Take gradient steps for $\mathcal{L}_C(\Lambda, \Phi)$ and $\mathcal{L}_D(\Lambda, \Psi)$;
  end
  `// Domain adversary`
  for *each batch sampled from* $\mathcal{D}_S^u$ *and* $\mathcal{D}_T^u$ **do**
    a) Compute $\mathcal{L}_D(\Lambda, \Psi)$
    b) Take a gradient step for $\mathcal{L}_D(\Lambda, \Psi)$;
  end
until *convergence*;

In each iteration, we do three kinds of gradient updates to account for the three different loss components. First, we do an epoch over all the training instances updating the parameters for the semi-supervised loss, then we do an epoch over the labeled instances in the source domain, each time updating the parameters for the supervised and the domain adversary losses. Finally, we do an epoch over the unlabeled instances in the two domains to account for the domain adversary loss.

The main challenge in adversarial training is to balance the competing components of the network. If one component becomes smarter than the other, its loss to the shared layer becomes useless, and the training fails to converge (Arjovsky et al., 2017). Equivalently, if one component becomes weaker, its loss overwhelms that of the other, causing the training to fail. In our experiments, we observed the domain discriminator is weaker than the rest of the network. This could be due to the noisy nature of tweets, which makes the job for the domain discriminator harder. To balance the components, we would want the error signals from the discriminator to be fairly weak, also we would want the supervised loss to have more impact than the semi-supervised loss. In our experiments, the weight of the domain adversary loss λ_d was fixed to $1e-8$, and the weight of the semi-supervised loss λ_g was fixed to $1e-2$. Other sophisticated weighting schemes have been proposed recently

[3] Tuning E on our task by backpropagation increased the training time immensely (3 days compared to 5 hours on a Tesla GPU) without any significant performance gain.

(Ganin et al., 2016; Arjovsky et al., 2017; Metz et al., 2016). It would be interesting to see how our model performs using these advanced tuning methods, which we leave as a future work.

2.5 Crisis Word Embedding

As mentioned, we used word embeddings that are pre-trained on a crisis dataset. To train the word-embedding model, we first pre-processed tweets collected using the AIDR system (Imran et al., 2014) during different events occurred between 2014 and 2016. In the preprocessing step, we lowercased the tweets and removed URLs, digit, time patterns, special characters, single character, username started with the @ symbol. After pre-processing, the resulting dataset contains about 364 million tweets and about 3 billion words.

There are several approaches to train word embeddings such as continuous bag-of-words (CBOW) and skip-gram models of wrod2vec (Mikolov et al., 2013), and Glove (Pennington et al., 2014). For our work, we trained the CBOW model from word2vec. While training CBOW, we filtered out words with a frequency less than or equal to 5, and we used a context window size of 5 and $k = 5$ negative samples. The resulting embedding model contains about 2 million words with vector dimensions of 300.

3 Experimental Settings

In this section, we describe our experimental settings – datasets used, settings of our models, compared baselines, and evaluation metrics.

3.1 Datasets

To conduct the experiment and evaluate our system, we used two real-world Twitter datasets collected during the *2015 Nepal earthquake* (NEQ) and the *2013 Queensland floods* (QFL). These datasets are comprised of millions of tweets collected through the Twitter streaming API[4] using event-specific keywords/hashtags.

To obtain the labeled examples for our task we employed paid workers from the Crowdflower[5] – a crowdsourcing platform. The annotation consists of two classes *relevant* and *non-relevant*. For the annotation, we randomly sampled 11,670 and 10,033 tweets from the Nepal earthquake and the Queensland floods datasets, respectively. Given a tweet, we asked crowdsourcing workers to assign the *"relevant"* label if the tweet conveys/reports information useful for crisis response such as a report of injured or dead people, some kind of infrastructure damage, urgent needs of affected people, donations requests or offers, otherwise assign the *"non-relevant"* label. We split the labeled data into 60% as training, 30% as test and 10% as development. Table 1 shows the resulting datasets with class-wise distributions. Data preprocessing was performed by following the same steps used to train the word2vec model (Subsection 2.5). In all the experiments, the classification task consists of two classes: *relevant* and *non-relevant*.

Dataset	Relevant	Non-relevant	Train	Dev	Test
NEQ	5,527	6,141	7,000	1,167	3,503
QFL	5,414	4,619	6,019	1,003	3,011

Table 1: Distribution of labeled datasets for Nepal earthquake (NEQ) and Queensland flood (QFL).

3.2 Model Settings and Baselines

In order to demonstrate the effectiveness of our joint learning approach, we performed a series of experiments. To understand the contribution of different network components, we performed an ablation study showing how the model performs as a semi-supervised model alone and as a domain adaptation model alone, and then we compare them with the combined model that incorporates all the components.

3.2.1 Settings for Semi-supervised Learning

As a baseline for the semi-supervised experiments, we used the self-training approach (Scudder, 1965). For this purpose, we first trained a supervised model using the CNN architecture (i.e., shared components followed by the supervised part in Figure 1). The trained model was then used to automatically label the unlabeled data. Instances with a classifier confidence score ≥ 0.75 were then used to retrain a new model.

Next, we run experiments using our graph-based semi-supervised approach (i.e., shared components followed by the supervised and semi-supervised parts in Figure 1), which exploits unlabeled data. For reducing the computational cost, we randomly selected $50K$ unlabeled instances from the same domain. For our semi-supervised setting, one of the main goals was to understand how much labeled data is sufficient to obtain a

[4]https://dev.twitter.com/streaming/overview
[5]http://crowdflower.com

reasonable result. Therefore, we experimented our system by incrementally adding batches of instances, such as 100, 500, 2000, 5000, and all instances from the training set. Such an understanding can help us design the model at the onset of a crisis event with sufficient amount of labeled data. To demonstrate that the semi-supervised approach outperforms the supervised baseline, we run supervised experiments using the same number of labeled instances. In the supervised setting, only $\mathbf{z}_c$ activations in Figure 1 are used for classification.

3.2.2 Settings for Domain Adaptation

To set a baseline for the domain adaptation experiments, we train a CNN model (i.e., shared components followed by the supervised part in Figure 1) on one event (source) and test it on another event (target). We call this as *transfer baseline*.

To assess the performance of our domain adaptation technique alone, we exclude the semi-supervised component from the network. We train and evaluate models with this network configuration using different source and target domains.

Finally, we integrate all the components of the network as shown in Figure 1 and run domain adaptation experiments using different source and target domains. In all our domain adaptation experiments, we only use unlabeled instances from the target domain. In domain adaption literature, this is known as *unsupervised* adaptation.

3.2.3 Training Settings

We use 100, 150, and 200 filters each having the window size of 2, 3, and 4, respectively, and pooling length of 2, 3, and 4, respectively. We do not tune these hyperparameters in any experimental setting since the goal was to have an end-to-end comparison with the same hyperparameter setting and understand whether our approach can outperform the baselines or not. Furthermore, we do not filter out any vocabulary item in any settings.

As mentioned before in Subsection 2.4, we used AdaDelta (Zeiler, 2012) to update the model parameters in each SGD step. The learning rate was set to 0.1 when optimizing on the classification loss and to 0.001 when optimizing on the semi-supervised loss. The learning rate for domain adversarial training was set to 1.0. The maximum number of epochs was set to 200, and dropout rate of 0.02 was used to avoid overfitting (Srivastava et al., 2014). We used validation-based *early stopping* using the F-measure with a patience of 25, i.e., we stop training if the score does not increase for 25 consecutive epochs.

Experiments	AUC	P	R	F1
NEPAL EARTHQUAKE				
Supervised	61.22	62.42	62.31	60.89
Semi-supervised (Self-training)	61.15	61.53	61.53	61.26
Semi-supervised (Graph-based)	64.81	64.58	64.63	65.11
QUEENSLAND FLOODS				
Supervised	80.14	80.08	80.16	80.16
Semi-supervised (Self-training)	81.04	80.78	80.84	81.08
Semi-supervised (Graph-based)	92.20	92.60	94.49	93.54

Table 2: Results using supervised, self-training, and graph-based semi-supervised approaches in terms of Weighted average AUC, precision (P), recall (R) and F-measure (F1).

3.2.4 Evaluation Metrics

To measure the performance of the trained models using different approaches described above, we use weighted average precision, recall, F-measure, and Area Under ROC-Curve (AUC), which are standard evaluation measures in the NLP and machine learning communities. The rationale behind choosing the weighted metric is that it takes into account the class imbalance problem.

4 Results and Discussion

In this section, we present the experimental results and discuss our main findings.

4.1 Semi-supervised Learning

In Table 2, we present the results obtained from the supervised, self-training based semi-supervised, and our graph-based semi-supervised experiments for the both datasets. It can be clearly observed that the graph-based semi-supervised approach outperforms the two baselines – supervised and self-training based semi-supervised. Specifically, the graph-based approach shows 4% to 13% absolute improvements in terms of F1 scores for the Nepal and Queensland datasets, respectively.

To determine how the semi-supervised approach performs in the early hours of an event when only fewer labeled instances are available, we mimic a batch-wise (not to be confused with minibatch in SGD) learning setting. In Table 3, we present the results using different batch sizes – 100, 500, 1,000, 2,000, and all labels.

From the results, we observe that models' performance improve as we include more labeled data

Exp.	100	500	1000	2000	All L
NEPAL EARTHQUAKE					
L	43.63	52.89	56.37	60.11	60.89
L+50kU	52.32	59.95	61.89	64.05	65.11
QUEENSLAND FLOOD					
L	48.97	76.62	80.62	79.16	80.16
L+∼21kU	75.08	85.54	89.08	91.54	93.54

Table 3: Weighted average F-measure for the graph-based semi-supervised settings using different batch sizes. *L* refers to labeled data, *U* refers to unlabeled data, *All L* refers to all labeled instances for that particular dataset.

— from 43.63 to 60.89 for NEQ and from 48.97 to 80.16 for QFL in the case of labeled only (***L***). When we compare supervised vs. semi-supervised (***L*** vs. ***L+U***), we observe significant improvements in F1 scores for the semi-supervised model for all batches over the two datasets. As we include unlabeled instances with labeled instances from the same event, performance significantly improves in each experimental setting giving 5% to 26% absolute improvements over the supervised models. These improvements demonstrate the effectiveness of our approach. We also notice that our semi-supervised approach can perform above 90% depending on the event. Specifically, major improvements are observed from batch size 100 to 1,000, however, after that the performance improvements are comparatively minor. The results obtained using batch sizes 500 and 1,000 are reasonably in the acceptable range when labeled and unlabeled instances are combined (i.e., *L+50kU* for Nepal and *L+∼21kU* for Queensland), which is also a reasonable number of training examples to obtain at the onset of an event.

4.2 Domain Adaptation

In Table 4, we present domain adaptation results. The first block shows event-specific (i.e., train and test on the same event) results for the supervised CNN model. These results set the upper bound for our domain adaptation methods. The *transfer* baselines are shown in the next block, where we train a CNN model in one domain and test it on a different domain. Then, the third block shows the results for the domain adversarial approach without the semi-supervised loss. These results show the importance of domain adversarial component. After that, the fourth block presents the performance of the model trained with graph embedding without domain adaptation to show the importance of semi-supervised learning. The final block present the results for the complete model that includes all the loss components.

Source	Target	AUC	P	R	F1
IN-DOMAIN SUPERVISED MODEL					
Nepal	**Nepal**	61.22	62.42	62.31	60.89
Queensland	**Queensland**	80.14	80.08	80.16	80.16
TRANSFER BASELINES					
Nepal	**Queensland**	58.99	59.62	60.03	59.10
Queensland	**Nepal**	54.86	56.00	56.21	53.63
DOMAIN ADVERSARIAL					
Nepal	**Queensland**	60.15	60.62	60.71	60.94
Queensland	**Nepal**	57.63	58.05	58.05	57.79
GRAPH EMBEDDING WITHOUT DOMAIN ADVERSARIAL					
Nepal	**Queensland**	60.38	60.86	60.22	60.54
Queensland	**Nepal**	54.60	54.58	55.00	54.79
GRAPH EMBEDDING WITH DOMAIN ADVERSARIAL					
Nepal	**Queensland**	66.49	67.48	65.90	65.92
Queensland	**Nepal**	58.81	58.63	59.00	59.05

Table 4: Domain adaptation experimental results. Weighted average AUC, precision (P), recall (R) and F-measure (F1).

The results with domain adversarial training show improvements across both events – from 1.8% to 4.1% absolute gains in F1. These results attest that adversarial training is an effective approach to induce domain invariant features in the internal representation as shown previously by Ganin et al. (2016).

Finally, when we do both semi-supervised learning and unsupervised domain adaptation, we get further improvements in F1 scores ranging from 5% to 7% absolute gains. From these improvements, we can conclude that domain adaptation with adversarial training along with graph-based semi-supervised learning is an effective method to leverage unlabeled and labeled data from a different domain.

Note that for our domain adaptation methods, we only use unlabeled data from the target domain. Hence, we foresee future improvements of this approach by utilizing a small amount of target domain labeled data.

5 Related Work

Two lines of research are directly related to our work: (*i*) semi-supervised learning and (*ii*) domain adaptation. Several models have been proposed for semi-supervised learning. The earliest approach is self-training (Scudder, 1965), in

which a trained model is first used to label unlabeled data instances followed by the model retraining with the most confident predicted labeled instances. The co-training (Mitchell, 1999) approach assumes that features can be split into two sets and each subset is then used to train a classifier with an assumption that the two sets are conditionally independent. Then each classifier classifies the unlabeled data, and then most confident data instances are used to re-train the other classifier, this process repeats multiple times.

In the graph-based semi-supervised approach, nodes in a graph represent labeled and unlabeled instances and edge weights represent the similarity between them. The structural information encoded in the graph is then used to regularize a model (Zhu, 2005). There are two paradigms in semi-supervised learning: 1) inductive – learning a function with which predictions can be made on unobserved instances, 2) transductive – no explicit function is learned and predictions can only be made on observed instances. As mentioned before, inductive semi-supervised learning is preferable over the transductive approach since it avoids building the graph each time it needs to infer the labels for the unlabeled instances.

In our work, we use a graph-based inductive deep learning approach proposed by Yang et al. (2016) to learn features in a deep learning model by predicting contextual (i.e., neighboring) nodes in the graph. However, our approach is different from Yang et al. (2016) in several ways. First, we construct the graph by computing the distance between tweets based on word embeddings. Second, instead of using count-based features, we use a convolutional neural network (CNN) to compose high-level features from the distributed representation of the words in a tweet. Finally, for context prediction, instead of performing a random walk, we select nodes based on their similarity in the graph. Similar similarity-based graph has shown impressive results in learning sentence representations (Saha et al., 2017).

In the literature, the proposed approaches for domain adaptation include supervised, semi-supervised and unsupervised. It also varies from linear kernelized approach (Blitzer et al., 2006) to non-linear deep neural network techniques (Glorot et al., 2011; Ganin et al., 2016). One direction of research is to focus on feature space distribution matching by reweighting the samples from the source domain (Gong et al., 2013) to map source into target. The overall idea is to learn a good feature representation that is invariant across domains. In the deep learning paradigm, Glorot et al. (Glorot et al., 2011) used Stacked Denoising Auto-Encoders (SDAs) for domain adaptation. SDAs learn a robust feature representation, which is artificially corrupted with small Gaussian noise. *Adversarial training* of neural networks has shown big impact recently, especially in areas such as computer vision, where generative unsupervised models have proved capable of synthesizing new images (Goodfellow et al., 2014; Radford et al., 2015; Makhzani et al., 2015). Ganin et al. (2016) proposed domain adversarial neural networks (DANN) to learn discriminative but at the same time domain-invariant representations, with domain adaptation as a target. We extend this work by combining with semi-supervised graph embedding for unsupervised domain adaptation.

In a recent work, Kipf and Welling (2016) present CNN applied directly on graph-structured datasets - citation networks and on a knowledge graph dataset. Their study demonstrate that graph convolution network for semi-supervised classification performs better compared to other graph based approaches.

6 Conclusions

In this paper, we presented a deep learning framework that performs domain adaptation with adversarial training and graph-based semi-supervised learning to leverage labeled and unlabeled data from related events. We use a convolutional neural network to compose high-level representation from the input, which is then passed to three components that perform supervised training, semi-supervised learning and domain adversarial training. For domain adaptation, we considered a scenario, where we have only unlabeled data in the target event. Our evaluation on two crisis-related tweet datasets demonstrates that by combining domain adversarial training with semi-supervised learning, our model gives significant improvements over their respective baselines. We have also presented results of batch-wise incremental training of the graph-based semi-supervised approach and show approximation regarding the number of labeled examples required to get an acceptable performance at the onset of an event.

References

Martín Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. *CoRR* abs/1701.07875. http://arxiv.org/abs/1701.07875.

Yoshua Bengio and Xavier Glorot. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. of the 13th Intl. Conference on Artificial Intelligence and Statistics*. Sardinia, Italy, AISTATS '10, pages 249–256.

Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18(9):509–517.

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proc. of EMNLP*. ACL, pages 120–128.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of MLR* 17(59):1–35.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. pages 513–520.

Boqing Gong, Kristen Grauman, and Fei Sha. 2013. Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In *ICML (1)*. pages 222–230.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., pages 2672–2680.

Muhammad Imran, Carlos Castillo, Fernando Diaz, and Sarah Vieweg. 2015. Processing social media messages in mass emergency: A survey. *ACM Computing Surveys (CSUR)* 47(4):67.

Muhammad Imran, Carlos Castillo, Ji Lucas, Patrick Meier, and Sarah Vieweg. 2014. AIDR: Artificial intelligence for disaster response. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, pages 159–162.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188* .

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* .

Qing Lu and Lise Getoor. 2003. Link-based classification. In *Proc. of ICML*.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. 2015. Adversarial autoencoders. *CoRR* abs/1511.05644. http://arxiv.org/abs/1511.05644.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2016. Unrolled generative adversarial networks. *CoRR* abs/1611.02163. http://arxiv.org/abs/1611.02163.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations*. Available as arXiv preprint arXiv:1301.3781.

Tom Mitchell. 1999. The role of unlabeled data in supervised learning. In *Proceedings of the sixth international colloquium on cognitive science*. Citeseer, pages 2–11.

Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36(11):2227–2240.

Dat Nguyen, Kamela Ali Al Mannai, Shafiq Joty, Hassan Sajjad, Muhammad Imran, and Prasenjit Mitra. 2017. Robust classification of crisis-related data on social networks using convolutional neural networks. In *International AAAI Conference on Web and Social Media*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Robert Power, Bella Robinson, and David Ratcliffe. 2013. Finding fires with twitter. In *Proceedings of the Australasian Language Technology Association Workshop 2013 (ALTA 2013)*. pages 80–89.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR* abs/1511.06434. http://arxiv.org/abs/1511.06434.

Tanay Saha, Shafiq Joty, Naeemul Hassan, and Mohammad Hasan. 2017. Regularized and retrofitted models for learning sentence representation with context. In *CIKM*. ACM, Singapore, pages 547–556.

H Scudder. 1965. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory* 11(3):363–371.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Michael Steinbach, George Karypis, Vipin Kumar, et al. 2000. A comparison of document clustering techniques. In *KDD workshop on text mining*. Boston, volume 400, pages 525–526.

István Varga, Motoki Sano, Kentaro Torisawa, Chikara Hashimoto, Kiyonori Ohtake, Takao Kawai, Jong-Hoon Oh, and Stijn De Saeger. 2013. Aid is out there: Looking for help from tweets during a large scale disaster. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1619–1629. http://www.aclweb.org/anthology/P13-1159.

Sarah Vieweg, Carlos Castillo, and Muhammad Imran. 2014. Integrating social media communications into the rapid assessment of sudden onset disasters. In *International Conference on Social Informatics*. Springer, pages 444–461.

Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* .

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* .

Xiaojin Zhu. 2005. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison.