Solar: Science of Entity Loss Attribution

Anshuman Mourya
Amazon
mouryaan@amazon.com

Prateek Sircar
Amazon
sircarp@amazon.com

Anirban Majumder
Amazon
majumda@amazon.com

Deepak Gupta
Amazon
dgupt@amazon.com

ABSTRACT

The ability to accurately pinpoint the location of an event (e.g. loss, fault or bug) is of fundamental requirement in many systems. While we have state-of-the-art models to predict likelihood of an outcome, being able to pinpoint to the entity responsible for the outcome is also important. For example, in an e-commerce setup, a lost package detection system needs to infer the reason or location

(delivery station, sort center, trucks) in case of a missing item, a network management system would like to diagnose nodes that are faulty based on end-end packet flow traces or a compiler needs to point out the exact location of a code that is erroneous. In this paper, we present an Attention based neural architecture for entity localization to accurately pinpoint the location of package loss in delivery network and bugs in erroneous programs. Our model performs well in scenarios where there is no annotation / ground truth for entities for localization. It can also adapt itself if annotations / ground truth is available for even a subset of entities by leveraging semi-supervision. The core of our model is a ladder-style architecture that helps us achieve state-of-the-art performance in both entity localization and detection. Further, to show the generality of our approach, we demonstrate its performance on a bug localization task for software programs. On a publicly available data-set, our solution outperforms the state-of-the-art technique by a significant margin.

CCS CONCEPTS

• **Computing methodologies** → **Semi-supervised learning settings**; **Neural networks**.

KEYWORDS

semi-supervised learning, neural networks, attention

ACM Reference Format:
Anshuman Mourya, Prateek Sircar, Anirban Majumder, and Deepak Gupta. 2022. Solar: Science of Entity Loss Attribution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539087

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539087

1 INTRODUCTION

Large-scale industrial systems involve well-orchestrated interplay between many automated components, the complexity of which evolves as the components get refined using the collected data. With the growing complexity, not only the system becomes susceptible to faults, break-downs and losses, it also presents a challenge to trace the issue to its root-cause and gain an in-depth understanding of such behavior. For example, in transportation and logistics industry, millions of packages are delivered across the globe daily. Yet a small fraction of them get lost for various reasons (e.g. non-compliance of standard operating principles), resulting in significant business loss and customer frustration. Manual inspection of each and every package is infeasible due to the sheer scale at which the system operates, making root-cause analysis a challenging problem.

On the other hand, inter-operability issues in a large-scale distributed system can result in network-wide disruption of services. In recent times, a DNS configuration error caused an outage of Facebook, WhatsApp, Instagram and Messenger services to millions of users worldwide[1]. To prevent such outages, one would like to localize the fault and resolve the issue quickly. Fault localization is challenging due to the large number of inter-connected entities (network elements, compute resources, databases, etc.) involved. Yet another parallel can be drawn from the objective of bug localization in software programs: many large-scale software systems are shipped with bugs which cause the system to perform incorrectly or in an unexpected way. Automatically localizing software bugs will help improve developer efficiency and the eventual quality of the software.

The common underlying theme is a notion of *entities* (e.g., delivery station, sort center, trucks etc for shipment loss, individual line of codes for bug localization and inter-connected devices for fault localization) and the definition of an *event* signifying the intended purpose of the system: shipment delivery, http connections and program execution. A fault corresponds to an anomalous event (e.g. shipment loss) which we need to attribute to the responsible entities. We refer to this problem as *Entity Loss Localization*. We use the terms localization and attribution interchangeably throughout this paper.

In this paper, we address the entity localization problem in the context of shipment losses in the delivery network of a major e-commerce platform. From the logistics perspective, when a customer places an order, the package gets shipped from the nearest warehouse and goes through various facilities to reach the delivery location. Similarly, for customer returns, the delivery agent

[1] https://www.nytimes.com/2021/10/04/technology/facebook-down.html

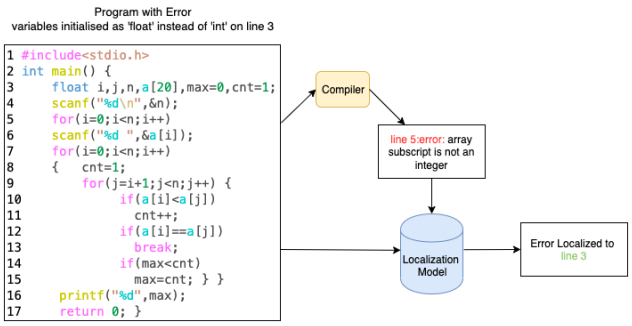

Figure 1: An example of bug localization in software programs. The compiler reports an error on line 5 whereas the bug is introduced in line 3 (best viewed in color).

picks up the item from the customer's doorstep and returns it to the warehouse. A small fraction of these packages get lost during transit. Pinpointing the exact location of loss is challenging for multiple reasons: (1) manual screening of every shipment is infeasible due to large number of shipments and entities (facilities, trucks, etc.) involved, (2) while there exists automated tracking mechanism for packages, the system is not 100% accurate and may give incorrect readings, e.g., due to interference of other packages in close proximity.

We present a machine learning based framework for entity loss localization. Specifically, we develop a novel attention based neural architecture (referred to as Science of Entity Loss Attribution or Solar) to jointly model the task of loss prediction and localization. We run extensive experiments (both offline and online) to demonstrate the superiority of Solar as compared to incumbent policies and other baselines.

We make a distinction between loss localization and loss prediction. Loss prediction involves estimating the probability of loss of a shipment on the basis of which one can intervene on high propensity packages. However, loss prediction systems can't pinpoint the loss location on fine-grained entities (delivery station, trucks etc) resulting in sub-optimal interventions. Solar fills in this gap by providing entity-grained loss location for each shipment, which opens up new possibilities to target precise interventions such as establishing inspection mechanisms in risky delivery station, sort center or trucks. We further note that Solar can be both preventive i.e. before the package has shipped, as well as reactive i.e. for packages that are already lost. We highlight the following contributions,

(1) We introduce the novel problem of loss localization. Conventional loss prevention models estimate the likelihood of loss. We highlight that with fine-grained loss localization, we can be extremely surgical in the selection of treatments leading to business waste reduction.
(2) Loss localization is challenging due to the lack of large scale ground-truth data. We propose Solar , a novel attention-based semi-supervised model to localize package loss at the entity level. The model leverages fine-grained information regarding entities (delivery station, sort center, trucks), and shipment level features to estimate the package loss probabilities. Through offline experiments, we demonstrate that our algorithm achieves considerable improvement in localization accuracy over existing baselines.
(3) To demonstrate the generality of our approach, we apply Solar to the problem of bug localization in software programs on a public data-set. Offline results indicate that our algorithms lead to as high as 1% lift in localization accuracy as compared to the state-of-the-art technique for bug localization.

After running an A/B test on a popular e-commerce company in an emerging marketplace, Solar achieved cost savings in the order of 10 basis points of revenue.

2 RELATED WORKS

Loss localization can be thought of as a distant variant of Multiple Instance Learning (MIL) [4, 6, 12, 22]. MIL is a weakly supervised class of learning algorithms where the learner receives a set of labeled *bag* of examples instead of individual samples. A bag is labeled positive if there is at least one positive sample in the bag. If all samples are from the negative class, the bag is labeled negative. The connection is as follows - delivered shipments correspond to negative bags in the MIL set-up. For a lost package, at least one of the entities will be responsible, which is akin to a positive bag. However, there is one fundamental difference between MIL and our set-up: in MIL, the label of an example is unknown but fixed whereas, the label of an entity is conditioned on the shipment i.e. the same entity can be good for one shipment but may not be for others. This poses a challenge, as the class labels of the samples are noisy (or conditioned on contextual information) and requires new algorithms to be developed.

There is some recent work [20, 27, 34, 38] on learning deep representations of set structured data. In contrast to the traditional approach of working with fixed dimensional data, the goal here is to learn representations of a variable number of objects. The representations are used to drive various tasks, like estimating population statistics, anomaly detection, etc.

Attention mechanism [2] introduced first for the machine translation task generates conditional distribution over input entities. This distribution is used to obtain a weighted representation of the input. Attention mechanism is now a predominant concept in many of the NLP [9], computer vision [32] and speech processing task [7]. It dynamically learns to pay attention to different part of input based on its relevance to certain task such as classification [8, 14], machine translation [39], summarization [33],object detection [40] etc. We have used attention mechanism to obtain a likelihood of entity responsible for loss in the system.

Semi-supervised Learning (SSL) methods are new, exciting much more applicable to real world applications, as it utilises only a few labeled training examples along with a large number of unlabeled training examples. SSL approaches are applied when data holds a certain assumptions [5], namely Smoothness assumption, Cluster assumption and Manifold assumption. There are well known SSL methods such as Proxy Label methods [1, 19, 28] which utilises model trained on labelled instances to generate labels for unlabelled

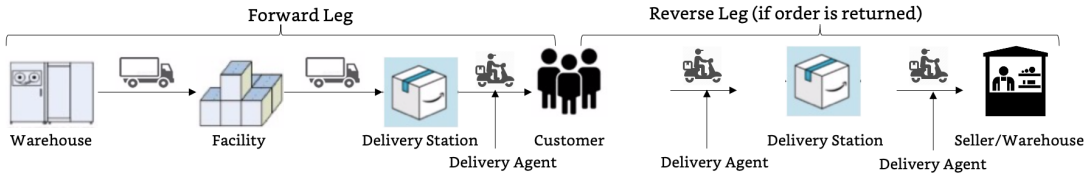

Figure 2: Figure showing the different steps and entities involved in delivery life-cycle. SOLAR model consumes information from all these stages.

set based on certain heuristics, Generative Models [16, 23] which learn to generate data from certain data distribution can learn to features that can be transferred to another supervised task, Graphical Models [17, 36] where data points are considered as nodes of the graph and information is propagated between nodes based on the similarity of two nodes, and Consistency Regularization [18, 25, 30] where a model is trained to obtain consistent output with an unlabeled example and its perturbed version. We have extended SOLAR model with consistency regularization technique to utilise majority of unlabelled examples for the package loss localization task.

Another related line of work is the research on multi-touch attribution for online advertising [26, 35]. The goal is to attribute the ad revenue to different channels in proportion to their impact on customer conversion. However, as the ground-truth attribution data is never available, these techniques are unsupervised in nature and can not be extended to semi-supervised setting for our use-case.

3 BACKGROUND

In Figure 2, we show the journey of an order from warehouse to customer doorstep and back to warehouse, in case of a return. The shipment can go missing at any of point of transit. The goal is to localize the loss to entities who were in contact with the shipment. Features related to these entities form the input to our model. Our localization algorithm is oblivious to the semantic of the entities. Depending on the intervention applied, one can use either coarse-grained (e.g. replacing facilities and trucks by a single network entity) or fine-grained entities without changing the underlying algorithm.

Problem Definition Let $\mathcal{E}$ be the universe of entities. Each shipment s defines a subset $\mathcal{E}_s \in \mathcal{E}$ of entities (possibly of variable length) who has been in contact with the shipment. Let $y_s \in \{0, 1\}$ be the target variable for the package. Let $\pi_s = (\pi_{s,i} : x_{s,i} \in \mathcal{E}_s)$ be a multinomial distribution over $\mathcal{E}_s$ where $\pi_{s,i}$ is the likelihood of entity $x_{s,i}$ being the reason for loss. *The goal of package loss localization is to infer the distribution π_s given the set of entities $\mathcal{E}_s$, and the shipment features λ_s.*

4 SOLAR MODEL

The objective of SOLAR has two main components (a) to be able to predict the final output - whether the shipment will lead to loss and (b) localization - identify the entity responsible for the loss. The first objective is a simple binary classification problem, which can be solved using a DNN, taking in all the features as input and predict an output. To address the second objective, we introduce attention weights to the model.

With attention mechanism, we train our deep learning model such that it pays attention to relevant entities of all the entities involved. We hypothesize that if we group features pertaining to an entity and pass it through a self-attention layer, the obtained attention weights can give each entity, a likelihood of being responsible for the output.

The SOLAR model comprises three layers of neural network - *input encoder*, *aggregator network* and *output layer*. The encoder layer maps high dimensional sparse feature vectors of entities to dense low dimensional representations. The low dimensional representations of a variable number of entities are combined by the aggregator network to a fixed dimensional representation of the input instance. This is picked by the output layer to predict the target variable. Below we describe each of these layers in detail. The architecture of the model is described in figure 3.

4.1 Input Encoder

For a given shipment s, let $\phi_{s,e}$ denote the feature vector corresponding to each entity $e \in \mathcal{E}_s$. These features could be ID of the entity (e.g. station ID of delivery station) represented as a learnable set of embedding parameters or contextual information about the entity and the action it performed on the shipment e.g. time of scan, percentage of lost shipments in contact with the entity in past 6 months etc. Note that the shipment feature vector λ_s captures shipment specific attributes (e.g. price, weight, category etc.). Each feature vector $\phi_{s,e}$ undergoes an encoding of the following nature

$$z_{s,e} = f_e(\phi_{s,e} \odot \lambda_s) \tag{1}$$

Here $\odot$ is the concatenation operation. The encoder function f_s first transforms the raw feature vectors through a batch normalization step followed by a single fully-connected layer with tanh non-linearity

$$\phi'_{s,e} = Batchnorm(\phi_{s,e}) \tag{2}$$

$$\phi''_{s,e} = tanh(W_{(e)}.\phi'_{s,e} + b_{(e)}) \tag{3}$$

Here $Batchnorm(.)$ [13] is a normalization step that learns two parameters (mean and standard deviation) and adjusts its input through scale and shift. This helps in improving the speed of convergence and stability during model training. $W_{(e)}$ and $b_{(e)}$ are entity-specific weights primarily aimed at bringing the entity-specific feature vectors from disparate feature spaces to a common representation. Shipment features λ_i are combined to output the latent

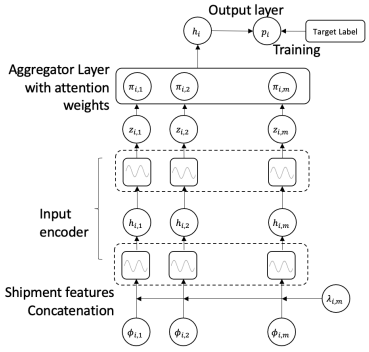

Figure 3: High Level Architecture of Solar model.

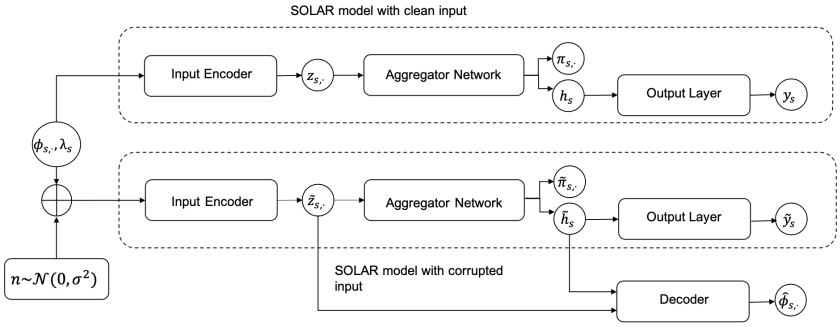

Figure 4: Ladder style architecture semi-supervised training of Solar model.

representation $z_{s,e}$ in the following manner:

$$h_{s,e} = (\phi''_{s,e} \odot \lambda_s) \tag{4}$$

$$h'_{s,e} = dropout(h_{s,e}) \tag{5}$$

$$z_{s,e} = tanh(W_h.h'_{s,e} + b_h) \tag{6}$$

Dropout [29] is a regularization technique that randomly sets a subset of input features to zero during the forward pass and back propagation. The latent representation $z_{s,e}$ is fed into aggregation layer to arrive at the final score.

4.2 Aggregator Network

Since we assume neither ordering nor dependency among entities, it is desirable that the likelihood function should be permutation invariant. Following the result of Zaheer et al [38], a permutation-invariant likelihood can be written as,

$$p(y_s \mid \mathcal{E}_s, \phi_{s,\cdot}, \lambda_s) = h\left(\sum_{e \in \mathcal{E}_s} f(\phi_{s,e}, \lambda_s)\right) \tag{7}$$

where f and h are functions implemented via Deep Neural Networks. Network f learns the encoding of the input features (refer to Section 4.1) which are then combined by the aggregator function h to learn an instance level representation.

In Solar we implement the aggregator function $h(\cdot)$ as a weighted combination of entity representations $\{z_{(s,e)} | e \in \mathcal{E}_s\}$ where the weights are learned via self-attention mechanism [21].

Let $\{\pi_{s,e} \mid e \in \mathcal{E}_s\}$ be a set of attention weights. Then the aggregation operation performed by this layer is given by

$$h_s = \sum_{e \in \mathcal{E}_s} \pi_{s,e} \cdot z_{s,e} \tag{8}$$

The attention weights are calculated as

$$\alpha_{s,e} = tanh(W_a \cdot z_{s,e}) \tag{9}$$

$$\beta_{s,e} = tanh(W'_a \cdot z_{s,e}) \tag{10}$$

$$\pi_{s,e} = \frac{\exp\left(w^T \cdot (\alpha_{s,e} \otimes \beta_{s,e})\right)}{\sum_{e'} \exp\left(w^T \cdot (\alpha_{s,e'} \otimes \beta_{s,e'})\right)} \tag{11}$$

where W_a, W'_a and w are learnable parameters and $\otimes$ is the element-wise multiplication operation. The attention weights help us focus on entities that are highly predictive of the target variable y_s than others. *Our interpretation is that for positive samples i.e. $y_s = 1$, high attention weights correspond to entities that are more likely to be responsible for the package loss.* As observed in our experiments (section 6), entities prone to losing the package (e.g. higher number of orders in past associated with package loss) tend to get higher attention weights.

4.3 Output Layer

In the final stage, the instance level representation h_s learned is passed through a stack of fully-connected layers with ReLU non-linearity. Further, we use residual connections and batch normalization for accelerated convergence of model training.

4.4 Semi-supervised Training of Solar

We have a modicum of ground-truth data-set on localization via investigations carried out by ground team. The dataset is small as it's limited by the capacity of human investigators. In order to make our model take advantage of the additional ground truth data, we propose semi-supervised learning of Solar via the Ladder architecture [25]. This way, the attention weights measuring loss localization, not only trains itself on input features but also takes information from the ground data wherever available. Consistency Regularization [3] is a recent line of work wherein the model utilizes unlabeled examples to enforce cluster assumption, i.e. the decision boundary should lie in the low-density region. We favor functions that give consistent outputs on similar data points, i.e. under the cluster assumption, a realistic perturbation to data should not change the predictions significantly.

Ladder Encoders Ladder network [25] takes a feed forward network and augments it with additional branches. Specifically, there is a clean encoder, a corrupted encoder, and a decoder. An input x passes through both the encoders, which are generally the same feed forward network but with Gaussian noise injected at each layer in the corrupted encoder after batch normalization. We thus obtain two predictions, y from clean encoder and $\tilde{y}$ from the corrupted encoder.

Ladder Decoder We use the representation $\widetilde{h}_s$ and $\widetilde{z}_{s,e}$ from the corrupted encoder to reconstruct the clean input. A network similar to Input Encoder is used with $[\widetilde{h}_s; \widetilde{z}_{s,e}]$ as input to obtain $\hat{\phi}_{s,e}$ as representation of each entity, with same dimensionality as that of $\phi_{s,e}$. We use denoised version of the decoder as well, wherein reconstructed activations of each layer in a FFNN or deeper network are utilised in the loss term.

At each training iteration, the input is passed through both Ladder Encoder networks. In the corrupted encoder, Gaussian noise is injected before the input encoding layer, producing two set of outputs, a clean prediction $[y, \pi_{s,e}]$ and a prediction based on corrupted input $[\widetilde{y}, \widetilde{\pi}_{s,e}]$. The output representation $\widetilde{h}_s$ is then fed into the decoder to reconstruct the clean input. The unsupervised training loss L_u is then computed as the sum of three losses - MSE between the output y of the clean encoder and $\widetilde{y}$, MSE between $\pi_{s,e}$ and $\widetilde{\pi}_{s,e}$, Reconstruction loss(MSE) between the clean encoder input $\phi_{s,e}$ and the reconstructed decoder predictions $\hat{\phi}_{s,e}$

$$L_u = \frac{\lambda_1}{|S|}\sum_s \|y_s - \widetilde{y}_s\|^2 + \lambda_2 \sum_s \frac{1}{|\mathcal{E}_s|}\sum_{e\in\mathcal{E}_s}\left\|\pi_{s,e} - \widetilde{\pi}_{s,e}\right\|^2 + \lambda_3 \sum_s \frac{1}{|\mathcal{E}_s|}\sum_{e\in\mathcal{E}_s}\left\|\phi_{s,e} - \hat{\phi}_{s,e}\right\|^2 \quad (12)$$

Supervised loss L_s has two components: (1) cross-entropy loss between clean encoder prediction y and ground truth prediction $\check{y}$, (2) if localization ground truth $\check{\pi}_{s,e}$ is available then cross entropy with attention weights from clean encoder $\pi_{s,e}$

$$L_s = -\frac{\lambda_4}{|S|}\sum_s [\check{y}_s \ln y_s + (1-\check{y}_s)\ln(1-y_s)] + \lambda_5 \sum_s \frac{1}{|\mathcal{E}_s|}\sum_{e\in\mathcal{E}_s}\check{\pi}_{s,e}\ln\pi_{s,e} \quad (13)$$

Here $\lambda_1 \cdots, \lambda_5$ are hyper-parameters for the Solar model. Finally, the total loss is computed as the sum of supervised and unsupervised losses which is used to backpropagate the gradients and learn model parameters.

$$L = L_s + L_u \quad (14)$$

5 EXTENDING ENTITY LOCALIZATION TO OTHER USE CASES: CODE BUG LOCALIZATION

We employ Semi-Supervised Learning over Graph to localise bugs in C/C++ programs using the Ladder Networks.

5.1 Problem Statement

Given a broken program $x = (x_1, ..., x_L)$ with L lines and compiler feedback message $f = (i_{err}, m_{err})$, where i_{err} denotes the reported line number, and m_{err} is a sequence of tokens constituting the error message, we identify the index of the erroneous line $k \in 1, ..., L$. Refer to the example presented in Fig.1. Line number reported by compiler feedback does not necessarily be identical to the actual erroneous line, thus we need Error Localization.

5.2 Ladder Networks for Bug localization

We adopt the architecture presented in [37] for localising and repairing the bugs. We provide a Ladder Network based extension on top of this model. We remove the error repair head from the network and use the remaining network as encoder in our architecture. Ladder networks use two encoders : one corrupted and one clean , and a decoder. Both the encoders have the same architecture as the localization model presented in [37]. Encoder consists of a line level bi-LSTM layer [11] for initial encoding, a graph attention layer [31] on program feedback graph (same construction as presented in [37]) constructed using tokens in lines and error message, another bi-LSTM layer for recontenxtualisation after graph attention and a program level bi-LSTM to generate scores for each line which is then used to predict line number using a MLP with softmax activation. Encoder has separate bi-LSTMs at each layer to encode code and compiler message. At each training iteration, the input x is passed through both encoders. In the corrupted encoder, Gaussian noise is injected at each layer after batch normalization, producing two outputs, a clean prediction y and a prediction based on corrupted activations $\widetilde{y}$. The output $\widetilde{y}$ is then fed into the decoder to reconstruct the uncorrupted input and the clean hidden activations. Ladder Decoder is a three layered network similar to the initial three layers of Ladder encoders (bi-LSTM encoding, graph attention and recontextualisation using another bi-LSTM), constructed to model the inverse of Encoders. All the three components of the architecture doesn't share any weights. Figure 5 has more details on the architecture.

Model Training At each training iteration, the input is passed through both Ladder Encoder networks. In the corrupted encoder, Gaussian noise is injected before LSTM layer (after Batch normalisation), producing two set of outputs, a clean prediction $y \in \mathbb{R}^{\mathbb{L}}$ and a prediction based on corrupted input $\widetilde{y} \in \mathbb{R}^{\mathbb{L}}$ where L is the number of lines in program . We also use the loss between clean encoder input representation $[x; m_{err}]$ and reconstructed decoder output $[x^d; m^d_{err}]$. The unsupervised training loss L_u is then computed as the sum of two losses - MSE between the output y of the clean encoder and $\widetilde{y}$ of corrupted encoder and the reconstruction error

$$L_u = \lambda_1 \frac{1}{|S|}\{\sum_s \sum_{k=1}^{L}\|y_{sk} - \widetilde{y}_{sk}\|^2 + \lambda_2 \sum_{i=1}^{L}\sum_{j=1}^{M}\left\|x_{ij} - x^d_{ij}\right\|^2 + \lambda_3 \sum_{j=1}^{M}\left\|m_{err} - m^d_{err}\right\|^2\} \quad (15)$$

where $|S|$ is the batch size, M is the maximum token in a line of code or message, L is the maximum number of lines in the code. Supervised loss L_s is the cross-entropy loss between clean encoder prediction y and ground truth prediction $\check{y}$ if localization ground truth is available

$$L_s = -\frac{1}{|S|}\sum_s \sum_{k=1}^{K}\check{y}_{sk}\ln y_{sk} \quad (16)$$

$$L = \lambda_4 L_s + L_u \quad (17)$$

Here $\lambda_1, .., \lambda_4$ are hyperparameters. Total loss is the weighted sum of supervised and unsupervised losses.

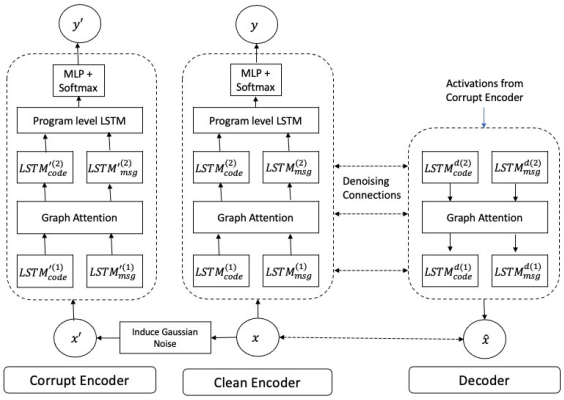

Figure 5: Ladder style architecture used for Bug localization

6 EXPERIMENTS

In this section, we present a convincing and thorough set of offline and online experiments to quantitatively evaluate the performance of our models and compare with incumbent techniques and other baselines. We first present our findings on the shipment loss data-set, followed by experiments on a public data-set on bug localization. For shipment loss localization, we find the likelihood of the overall loss and localize it between entities. For proprietary reasons, we would call the entities in consideration, entity A and entity B.

6.1 Baselines

We use the following baseline algorithms to compare with our proposed framework.

(1) Baseline 1: Here, we predict the likelihood of overall loss using an XGBoost model. For localization, we come up with simple risk features. We calculated all orders the entity was involved in a defined time period (say 1 year). We find the number of orders the entity was involved in which led to loss. The ratio of the 2 would be used as a proxy for localization score.

(2) Attention Model without Semi-supervision: We leverage the model described in sections 4.1, 4.2, 4.3. This is also our proposed final model in scenarios where there is no entity level ground truth data.

(3) Ladder Networks: We leverage the model described in sections 4.4. This is our final proposed model in presence of entity level ground truth data. We show results for ladder networks and ladder networks with denoising.

6.2 Experimental Setup

We use Pytorch [24] library to train our Solar model on loss of returned packages. Training was done on a single Nvidia V100 GPU. Batch size was set to be 250. Learning rate was set to be 0.001. We use ADAM [15] optimizer with parameters (beta1: 0.9, beta2: 0.999, epsilon: 10^{-8}). We used a weight decay of 0.95 for ADAM optimizer. We trained the model for 10 epochs.

6.3 Dataset

We take a randomly sampled data of returned orders for this experiment. Our training data and testing data has 2.5 million records each. Each row of the training data pertains to a returned order. For each order, we had IDs and some related features of two entities (anonymously referred to as 'Entity A' and 'Entity B'). Ground truth data on entities wherever they were available, were obtained from multiple investigations done on the entities in the past.

6.4 Results

We report increment in ROC-AUC and value recall at the top 2% and 10% of predicting loss over baseline 1. Localization for loss is limited to entities A and B. To measure the model performance of localization, we report increment in ROC-AUC and recall at top 2% and 10% of entity A and entity B risk score over baseline 1. We summarize the results of our experiment in Table 1. In the table, "value recall" refers to the recall of the total loss value. Entity results are measured against the existing ground truth data for corresponding entities. In all scenarios considered, our model achieves significant improvement over baselines in terms of AUC and model recall at the critical threshold. We can also observe that our Ladder Network performs better than the semi-supervised network for both loss localization and overall performance.

Table 1: Comparision of improvement in offline mertrics over baseline 1. Absolute numbers not shown due to confidentiality. A: Attention Model w/o Semi-supervision, B: Ladder Network, C: Ladder Network with denoising

Results	A	B	C
Package Loss AUC	+0.001	**+0.005**	+0.004
Package Loss top 2% value recall	+1.75%	**+5.53%**	+4.72%
Package Loss top 10% value recall	+0.17%	**+1.75%**	+1.32%
entity A risk AUC	+0.044	**+0.114**	+0.104
entity A risk top 2% recall	+37.71%	+50.48%	**+51.48%**
entity A risk top 10%recall	+0.67%	+12.19%	**+13.00%**
entity B risk AUC	+0.009	**+0.044**	+0.021
entity B risk top 2% recall	+0.80%	**+3.98%**	+3.78%
entity B risk top 10% recall	+4.42%	**+9.45%**	+4.82%

Ladder Network variants use five loss terms mentioned in the section 4.4. We measure the AUC drop compared to Ladder AUC numbers when we remove certain loss terms while training the model. We observe a 1.2% drop in package loss AUC, 4% drop in entity A risk AUC and 1% drop is entity B risk AUC after removing the unsupervised loss terms from the model training. When we removed two MSE loss terms contributing to consistency regularization, we observe a 0.2%, 6% and 1.8% drop in package loss, entity A risk and entity B risk AUC, respectively. With third ablation, we used only the cross entropy loss for package loss labels to train the model and no other loss terms, this gave us a drop of 1.4%, 9.6% and 1.1% in package loss, entity A risk and entity B risk AUC, respectively. As last ablation, we remove the cross entropy loss and MSE loss terms for attention weights, thus ending with just reconstruction loss, cross entropy for package loss and MSE for package loss. Under this last setting, we obtained a drop of 0.4%, 10% and 4.3% drop in package loss, entity A risk and entity B risk AUC, respectively. We conclude that contribution of each loss term while training is evident from these results, as we obtain best offline performance using all the fives loss terms together.

6.5 Evaluating Localization Accuracy

We performed another experiment to test the efficacy of the entity A localization score. We worked with teams which specialize in conducting manual investigation for entity A by looking at its past transactions and concluding whether there is any evidence to establish that the loss can be attributed to it. In the process, the model would run on each returned order and estimate the loss propensity at order level and the loss localization score for the entity A. We generate two samples of returned orders:

(1) **Sample X**: orders with high loss propensity
(2) **Sample Y**: orders with high loss propensity and having entity A localization score above a certain threshold.

The samples are mixed and sent for assessment to investigators, who does not know whether the current order under investigation belongs to sample X or Y. Basis investigation, the entity for the orders are flagged as responsible / not responsible for the loss. We observed that percentage of entity A flagged as responsible for loss in Sample Y is **44%** more than those in Sample X. Note that further details on the experiment set-up and outcome are withheld due to confidentiality. The outcome suggests that incorporating SOLAR predicted localization information leads to significantly higher accuracy in a production set-up for running investigations.

6.6 Online Experiments for Shipment Loss Localization

To test the efficacy of the framework, we conducted two A/B tests, one for entity A and other for entity B. While the treatment group was evaluated basis the SOLAR model, we use legacy techniques for the control cohort. The SOLAR model is invoked whenever a return order is generated in the treatment cohort. The model predicts the likelihood of lost package using the overall model score and localizes the loss to either entity A or entity B. Action to be taken to prevent the loss depends on which entity got a high risk score. Details of the process can be seen in Figure 6. The model is retrained on the additional ground-truth data available post inspection and investigation. A/B test results indicate incremental cost savings

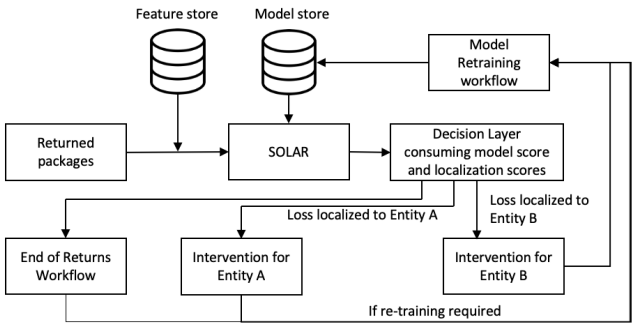

Figure 6: Shipment loss localization system leveraging predictions from SOLAR.

of the order of 10 basis points of revenue. The model has been deployed in production and is invoked on all return shipments for an emerging marketplace.

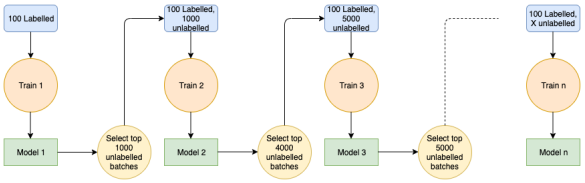

Figure 7: Multiple rounds of training with entropy based selection used at each training round.

6.7 Results for Bug Localization

In this section, we report the results on bug localization. This task differs from the lost package localization in a way that we don't have detection task. It is already known that the given code has an error (already detected to be erroneous) and the bug needs to be localised to a particular line of code. We perform two major experiments to establish the utility of our model:

- **Experiment 1 - Performance over baseline :** This experiment on Deepfix data [10] focus on establishing that ladder networks provide performance gain over the existing model for localization(baseline1) in [37]. Baseline1 is a supervised model trained using just the labelled examples. Ladder Network, on the other hand, uses unlabelled examples as well. With very small number of labelled examples, baseline1 outperforms ladder network, which is intuitive as Ladder predictions get perturbed due to unsupervised losses which are absent in loss function of baseline1.
- **Experiment 2 - Entropy Based Selection with multiple training rounds :** Our initial experiments on Bug localization proved that adding unlabelled examples randomly to the training data does not yield desired improvements over the model trained just with labelled examples. To expedite this, we employ an entropy based selection process that selects the unlabelled examples to use in the next round of model training. As depicted in 7, each successive round of model training include the same number of labelled examples but an incremental number of unlabelled examples. In each training round, we train a new model without considering the model from previous training round. Entropies are calculated for each batch on the basis of previous model training round and a specific number of batches with minimum entropy are shortlisted for next round of training. Entropy is defined as $-p_i log p_i$, where p_i is the predicted probability from the model for example i.

6.7.1 Dataset. We leverage the DeepFix dataset and Codeforce data which contains programming codes in C language that contained 34715 and 37466 programs respectively with an average program length of 25 lines. While few programs were working correctly, some were broken (do not compile). The broken programs are called raw test sets and may contain multiple errors. It was constructed originally for the task to repair them into ones that compile (full repair; evaluation metric is full repair rate). Our goal is to localise the line of error, rather than the correction. In the paper [37], authors have generated 50 corrupted versions of each program by the DrPerturb module. All the examples have ground

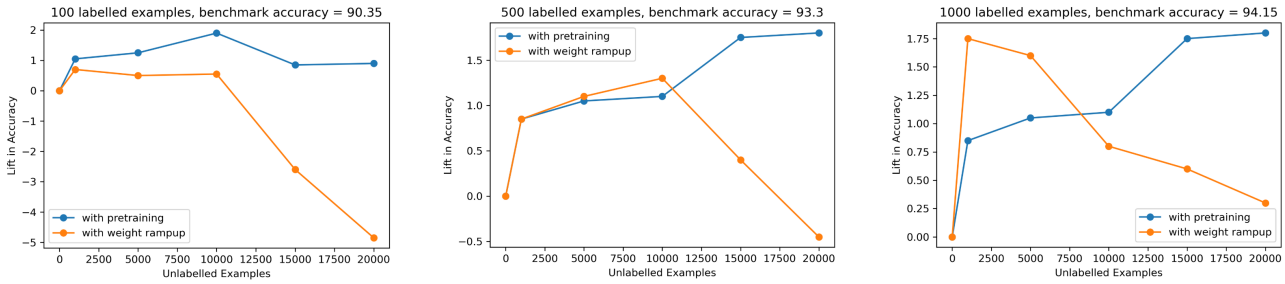

Figure 8: Accuracy improvement with entropy based selection for various configurations.

truth labels, we perform random sampling and select few examples to simulate the semi-supervised setting. The Dev set we use is standard from this paper, consisting of 2000 examples.

6.7.2 Testbed and Methodology. We observe that weights assigned to individual loss terms while training plays vital role. Models performed poorly if higher weights are assigned to unsupervised loss terms at the start of training. We experimented with some methods to address the problem of using variable weights throughout the training process. The maximum value for the unsupervised loss component was set to $w_{max}M/N$ where w_{max} is fixed in the first method and set variable in the second one. M is the number of labeled training data and N is the total training data (labelled + unlabeled). We devised two strategies : Pretraining and weight ramp up function. With a pretraining strategy, we use just labeled examples at the start of training till certain number of epochs and then introduce unlabeled examples into training. This made model training in the initial epochs much more stable. As second strategy, we experimented with using a weight ramp up function to gradually increase the contribution of unsupervised loss terms and finally capping it to a certain value [18]. We did this using a Gaussian ramp-up curve $\exp[-5(1-T)^2]$ where T is the training step.

We use Pytorch [24] library to extend implementation by [37] with our models. We use the optimal hyperparameters used in the paper for experimentation, with batch size of 25 and 50. Experiments were run on a single Nvidia V100 GPU using Adam Optimiser and initial learning rate of 0.0001. Hidden dimensions for LSTM layer, graph layer and output layer is set same as [37].

6.7.3 Results. We report localization accuracy as the evaluation metric for predicting the line number of the bug in a software program. We count a success for a test program if we correctly predict any one of the line of error in the program. As shown in Table 2, our semi-supervised localization achieves 44bps of lift in accuracy over state-of-the-art supervised techniques.We used 1870750 examples from Deepfix dataset for this experiment, which were 37415 programs with 50 corrupted versions of each. We establish that our model is winner in Experiment 1. For Experiment 2, we selected 100, 500 and 1000 supervised batches (batchsize=25) and incrementally built models with entropy based selection of unlabeled examples. We use 1000, 5000, 10000, 15000 and 20000 chosen incrementally in each model training round. We experimented with both the loss weighting methods mentioned in 6.7.2. Our baseline varies for each configuration of labeled examples. For each experiment, we begin

Table 2: Comparison of baseline model with Ladder Networks with varying number of labeled examples.

Percentage Supervision	Dev Acc on Baseline	Dev Acc on Ladder Networks	Lift
100	96.85	**97.5**	0.65
10	96.30	**96.35**	0.05
1	91.65	**92.60**	0.95
0.1	84.95	**85.05**	0.1

with a different number of labeled examples. Each baseline model is trained only on the specified number of labeled examples. In 8, we observe a continuous increase in lift with the first weighting method (involving pretraining) and some irregular behaviour with the second , when the number of unlabeled examples increases. Moreover, with second method, as the number of labelled example increases, the degradation in lift is slower. In case of 1000 labelled examples we get a positive lift on all the operating points.

7 CONCLUSION AND FUTURE WORK

The ability to accurately pinpoint the location of package loss is a fundamental requirement of many loss prevention initiatives across e-commerce companies. In this paper, we have presented a) semi-supervised localization algorithm and b) Ladder Networks. We also showcased that our modeling approach could be generalized for any fault localization algorithm by demonstrating results on Bug Localization. Through experiments and implementation details, we show that our model outperforms baselines by a significant margin across different loss buckets. Same framework has showed inspiring results on localizing the erroneous line in the code with the ladder-augmented semi supervised model. As a next step, work is presently ongoing for extending our algorithm to more complex network types or graphs. We believe this approach might give us improvement in tasks where such a complex structure exists.

REFERENCES

[1] Eric Arazo, Diego Ortego, Paul Albert, Noel E. O'Connor, and Kevin McGuinness. 2020. Pseudo-Labeling and Confirmation Bias in Deep Semi-Supervised Learning. (2020). https://openreview.net/forum?id=rJel41BtDH

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural Machine Translation by Jointly Learning to Align and Translate. (2016). arXiv:cs.CL/1409.0473

[3] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. 2019. MixMatch: A Holistic Approach to Semi-Supervised Learning. In *Advances in Neural Information Processing Systems*,

Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/1cd138d0499a68f4bb72bee04bbec2d7-Paper.pdf

[4] Marc-Andr Carbonneau, Veronika Cheplygina, Eric Granger, and Ghyslain Gagnon. 2018. Multiple Instance Learning. *Pattern Recogn.* 77, C (May 2018), 329–353. https://doi.org/10.1016/j.patcog.2017.10.009

[5] Olivier Chapelle, Bernhard Scholkopf, and Eds. A. Zien. 2009. Semi-Supervised Learning (Chapelle, O. et al., Eds.; 2006) [Book reviews]. *IEEE Transactions on Neural Networks* 20 (2009), 542–542.

[6] Veronika Cheplygina, David M.J. Tax, and Marco Loog. 2015. On Classification with Bags, Groups and Sets. *Pattern Recogn. Lett.* 59, C (July 2015), 11–17. https://doi.org/10.1016/j.patrec.2015.03.008

[7] KyungHyun Cho, Aaron C. Courville, and Yoshua Bengio. 2015. Describing Multimedia Content using Attention-based Encoder-Decoder Networks. *CoRR* abs/1507.01053 (2015). arXiv:1507.01053 http://arxiv.org/abs/1507.01053

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. https://openreview.net/forum?id=YicbFdNTTy

[9] Andrea Galassi, Marco Lippi, and Paolo Torroni. 2019. Attention, please! A Critical Review of Neural Attention Models in Natural Language Processing. *CoRR* abs/1902.02181 (2019). arXiv:1902.02181 http://arxiv.org/abs/1902.02181

[10] Rahul Gupta, Aditya Kanade, and Shirish Shevade. 2019. Neural Attribution for Semantic Bug-Localization in Student Programs. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/f29a179746902e331572c483c45e5086-Paper.pdf

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (nov 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[12] Judy Hoffman, Deepak Pathak, Trevor Darrell, and Kate Saenko. 2015. Detector Discovery in the Wild: Joint Multiple Instance and Representation Learning. In *Computer Vision and Pattern Recognition (CVPR)*.

[13] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 448–456. http://proceedings.mlr.press/v37/ioffe15.html

[14] Douwe Kiela, Changhan Wang, and Kyunghyun Cho. 2018. Dynamic Meta-Embeddings for Improved Sentence Representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. 1466–1477. https://doi.org/10.18653/v1/d18-1176

[15] D Kingma and J Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[16] Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling. 2014. Semi-Supervised Learning with Deep Generative Models. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 3581–3589.

[17] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016). arXiv:1609.02907 http://arxiv.org/abs/1609.02907

[18] Samuli Laine and Timo Aila. 2016. Temporal Ensembling for Semi-Supervised Learning. *CoRR* abs/1610.02242 (2016). arXiv:1610.02242 http://arxiv.org/abs/1610.02242

[19] Dong-Hyun Lee. 2013. Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks.

[20] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018. PointCNN: Convolution On X-Transformed Points. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 820–830. http://papers.nips.cc/paper/7362-pointcnn-convolution-on-x-transformed-points.pdf

[21] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A Structured Self-attentive Sentence Embedding. *ICLR* (2017).

[22] Amy McGovern and David Jensen. 2003. Identifying Predictive Structures in Relational Data Using Multiple Instance Learning. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML'03)*. AAAI Press, 528–535.

[23] Augustus Odena. 2016. Semi-Supervised Learning with Generative Adversarial Networks. (2016). arXiv:stat.ML/1606.01583

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[25] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. 2015. Semi-supervised Learning with Ladder Networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3546–3554. http://papers.nips.cc/paper/5947-semi-supervised-learning-with-ladder-networks.pdf

[26] Kan Ren, Yuchen Fang, Weinan Zhang, Shuhao Liu, Jiajun Li, Ya Zhang, Yong Yu, and Jun Wang. 2018. Learning Multi-Touch Conversion Attribution with Dual-Attention Mechanisms for Online Advertising *(CIKM '18)*. Association for Computing Machinery.

[27] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. 2017. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4967–4976. http://papers.nips.cc/paper/7082-a-simple-neural-network-module-for-relational-reasoning.pdf

[28] Weiwei Shi, Yihong Gong, Chris Ding, Zhiheng MaXiaoyu Tao, and Nanning Zheng. 2018. Transductive Semi-Supervised Deep Learning using Min-Max Features. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[29] Nitish Srivastava, Geoffrey 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, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[30] Antti Tarvainen and Harri Valpola. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 1195–1204. https://proceedings.neurips.cc/paper/2017/hash/68053af2923e00204c3ca7c6a3150cf7-Abstract.html

[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. 2018. Graph Attention Networks. *ArXiv* abs/1710.10903 (2018).

[32] Feng Wang and David M. J. Tax. 2016. Survey on the attention based RNN model and its applications in computer vision. *CoRR* abs/1601.06823 (2016). arXiv:1601.06823 http://arxiv.org/abs/1601.06823

[33] Song Xu, Haoran Li, Peng Yuan, Youzheng Wu, Xiaodong He, and Bowen Zhou. 2020. Self-Attention Guided Copy Mechanism for Abstractive Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. 1355–1362. https://doi.org/10.18653/v1/2020.acl-main.125

[34] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. 2018. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. *ECCV* (2018), 87–102.

[35] Dongdong Yang, Kevin Dyer, and Senzhang Wang. 2020. Interpretable Deep Learning Model for Online Multi-touch Attribution. *CoRR* abs/2004.00384 (2020). https://arxiv.org/pdf/2004.00384.pdf

[36] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. *CoRR* abs/1603.08861 (2016). arXiv:1603.08861 http://arxiv.org/abs/1603.08861

[37] Michihiro Yasunaga and Percy Liang. 2020. Graph-based, Self-Supervised Program Repair from Diagnostic Feedback. *ArXiv* abs/2005.10636 (2020).

[38] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 3391–3401. http://papers.nips.cc/paper/6931-deep-sets.pdf

[39] Biao Zhang, Deyi Xiong, and Jinsong Su. 2017. A GRU-Gated Attention Model for Neural Machine Translation. *CoRR* abs/1704.08430 (2017). arXiv:1704.08430 http://arxiv.org/abs/1704.08430

[40] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. 2021. Deformable DETR: Deformable Transformers for End-to-End Object Detection. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. https://openreview.net/forum?id=gZ9hCDWe6ke