Invernet: An Adversarial Attack Framework to Infer Downstream Context Distribution through Word Embedding Inversion

Ishrak Hayet

Submitted to the graduate degree program in Department of Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Committee members

Dr. Bo Luo, Chair

Dr. Zijun Yao, Co-chair

Dr. Fengjun Li

Dr. Alexandru Bardas

Date defended: April 26, 2022

The Thesis Committee for Ishrak Hayet certifies
that this is the approved version of the following thesis :

Invernet: An Adversarial Attack Framework to Infer Downstream Context Distribution through Word Embedding Inversion

____

Dr. Bo Luo, Chair

Date approved: April 26, 2022

Abstract

Word embedding has become a popular form of data representation that is used to train deep neural networks in many natural language processing tasks, such as machine translation, named entity recognition, information retrieval, etc. Through embedding, each word is represented as a dense vector which captures its semantic relationship with others, and can better empower machine learning models to achieve state-of-the-art performance. Due to the data and computation intensive nature of learning word embeddings from scratch, an affordable way is to borrow an existing general embedding trained on large-scale text corpora by third party (i.e., pre-training), and further specialize the embedding by training on downstream domain-specific dataset (i.e., fine-tuning). However, a privacy issue can rise during this process is that the adversarial parties who have the pre-train datasets may be able infer the key information such context distribution of downstream datasets by analyzing the fine-tuned embeddings.

In this study, we aim to propose an effective way to infer the context distribution (i.e., the words co-occurrence in downstream corpora revealing particular domain information) in order to demonstrate the above-mentioned privacy concerns. Specifically, we propose a focused selection method along with a novel model inversion architecture “Invernet” to invert word embeddings into the word-to-word context information of the fine-tuned dataset. We consider the popular word2vec models including CBOW, SkipGram, and GloVe algorithms with various unsupervised settings. We conduct extensive experimental study on two real-world news datasets: Antonio Gulli’s News Dataset from Hugging Face repository and a New York Times dataset from both quantitative and qualitative perspectives. Results show that “Invernet” has been able to achieve an average F1 score of 0.70 and an average AUC score of 0.79 in an attack scenario.

A concerning pattern from our experiments reveal that embedding models that are generally considered superior in different tasks tend to be more vulnerable to model inversion. Our results suggest that a significant amount of context distribution information from the downstream dataset can potentially leak if an attacker gets access to the pretrained and fine-tuned word embeddings. As a result, attacks using “Invernet” can jeopardize the privacy of the users whose data might have been used to fine-tune the word embedding model.

Acknowledgements

I would like to thank Dr. Luo and Dr. Yao for their valuable guidance and suggestion. Without their effective and helpful suggestions and guidance, this work would be impossible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Discriminative data representation learning is a critical factor in determining the success of Machine Learning models. In Natural Language Processing (NLP), many deep neural networks need the input words or documents to be represented in a form of numerical vectors which indicate the semantical distinction among information, so that the models can improve the performance of NLP tasks relying on these quality input. Different methods of converting vocabulary tokens to numbers have been discussed in literature [40]. For example, an intuitive way of converting categorical vocabulary to numerical vectors is to one-hot encode the tokens into a bitstring or commonly known as a bag of words representation [17]. However, one-hot encoding can easily result in high-dimensional representations and usually is not capable of accommodating unseen information. Another way of representation is using distributed but low-dimensional representation called word embedding [20], which has become successful and popular lately by capturing complex and subtle semantic difference of input words or documents [4].

In practice, well trained word embeddings demand large training corpus, sufficient training iterations, high computational cost and leaves a environmentally detrimental carbon footprint. E.g. in [38], the authors mention that training a BERT-base model emits around 1,500 lbs of carbon dioxide and costs around \$3,000 - \$12,000 to train. Ensuring such environmentally and financially costly training factors are usually infeasible or uneconomical for most small organizations and individuals. As a result, large platforms (e.g., Google cloud, Hugging Face) with computational capacity train word embeddings on large and generic datasets, and offer the pretrained embeddings through platforms like the Tensorflow Model Hub or the Hugging Face Model Repository [41]. With specific tasks and datasets in hand, other user entities can further refine the embeddings by training on their own datasets, while using pretrained embeddings as the input to generate their domain adapted downstream embeddings with much smaller corpus and cost. Such refinement process is known as fine-tuning and can be considered as the de facto standard practice of how word representations are commonly learnt in today's natural language processing tasks [32]. Examples of adopting such fine-tuning process are abundant in the biomedical sector [15], the financial sector [3] etc.

Often times, users of word embeddings are of the opinion that word embeddings are simply dense vector representations of words and that nothing much can be inferred about the training corpus from just a collection of numbers. So, most of the times, they do not exercise caution in keeping their word embeddings secure. In this paper, we study whether downstream word embeddings leak the context information of the downstream dataset when the fine-tuned word embeddings is available for adversarial parties. Specifically, under the situation when the adversarial parties like the pretrained embedding publishers are able to access the downstream embeddings, we study the methods of inferring word to word binary co-occurrence statistics of the downstream fine-tuning corpus, using the leaked fine-tuned embeddings and prior knowledge of the pretrained embeddings and datasets. In this work, we conduct four different experiments where the first one is a naive baseline that uses a logistic regression based inference model, the second one applies a motion based inference, the third one uses a stacked ensemble model and the fourth one i.e. "Invernet" uses an inference sampling approach along with a deep neural network to invert fine-tuned embeddings into the corresponding context.

Attacks on Machine Learning models have been studied extensively [28]. In [35], the authors train adversarial shadow models to carry out membership inference attacks against different commercial models. Feature based analyses of information leakage in machine learning models are discussed in [9], and [36]. In [9], the authors consider the owner of the downstream dataset as the attacker and perform a feature alignment based attack on the source dataset in a deep transfer learning setting. In [36], the authors try to infer the constituent words of a sentence given that they have access to a sentence embedding. The authors report an F1 score of 0.5-0.7 in [36]. However, our proposed research problem is different and novel in that we consider the owner of the source dataset to be the attacker and analyze information leakage of downstream dataset. We also consider the attack surface to be just the word embedding vectors of the source and the downstream dataset. Such an attack surface of word embedding vectors render our proposed problem and inversion methods generic across different word embedding models. We apply our method to word vectors that are generated by various major word embedding models like Word2Vec, and GloVe. Our methods successfully predict the neighboring words of a target word in the downstream dataset. Such context leakage can reveal critical information regarding the downstream dataset. We achieve an average F1 score of 0.70 and an average AUC score of 0.79 which is significant given the generic nature of the attack surface.

In one proposed approach, we hypothesize that the motion of a word vector can reveal contextual information of the dataset it is trained on. For this approach, we are required to store the state of individual word vectors at every training epoch. Storing epoch specific word vectors is feasible for the owner of the pretraining dataset because the owner of the pretraining dataset oversees the word embedding training process. However, even with access to the final form of the fine-tuned downstream embedding, the owner of the pretraining dataset might not get access to the fine-tuning process of the downstream embedding and hence she will not get access to the state of fine-tuned embeddings at each time step of the fine-tuning process. So, an alternative is to design a spatio-temporal encoder that can capture a latent representation of the temporal motion of the word vectors. We train this encoder to learn a latent motion encoding of the source embeddings based on only the initial and final form of the pretrained embeddings. Then, we transfer this encoder model to the downstream embeddings and try to approximate the latent motion encoding of the fine-tuned embeddings. In the final step of this method, we design a deep neural network that can learn the context information of a target word from the latent motion encoding of that word by training on the target word's context from the source dataset. Finally, we use this trained deep neural network to predict the target word's context in the downstream dataset.

In another approach, we develop an inversion framework "Invernet" for inverting downstream embeddings with a prior knowledge of the pretrained embeddings and the pretraining dataset. The primary components of the "Invernet" model architecture are a focused document selector, and a deep learning based inference model. The goal of the focused selector is to ensure that the selected samples from the dataset will not contain any noise. The inference model is designed as a learning problem where a deep model tries to learn the relation between the embeddings and the downstream context information. We compare the performance of our method with respect to how much context information of the downstream dataset is leaked in the inversion process. In the "Invernet" method, we divide the dataset into multiple sub-datasets and fine-tune the pretrained embeddings on the sub-datasets. Then we concatenate the pretrained embeddings and the downstream embeddings. We implement a series of fully-connected dense layers with self-attention and dropout layers which take the concatenated embeddings as input and gives the context information as output. After applying the "Invernet" method, we have found a considerable amount of context leakage within a specific window.

1.1 Contributions

The main contributions of our work are summarized as follows:

- Presenting a novel privacy problem of inverting fine-tuned embeddings into the contextual information of downstream dataset.
- Designing an inference attack model using deep learning techniques with multiple inference sub-sampling strategies to accurately predict contextual information of downstream datasets.
- Conducting comprehensive experimental study on two real-world datasets such as AG_News [16] and NYT Dataset [44]. Results clearly demonstrate the advantage of the proposed Invernet framework over all the baselines.

1.2 Thesis Organization

The thesis is organized into the following chapters:

- Chapter 1: Introduction - We introduce the problem and briefly mention our solution approaches. We motivate regarding why the problem is a threat to user privacy.

- Chapter 2: Problem Formulation - We formally define the problem and present an example of our goals

- Chapter 3: Background - In this chapter, we mention some notable relevant background work on Word Embeddings, Transfer Learning, and Privacy Issues with Word Embeddings

- Chapter 4: Methodology - We explain in detail our solution approaches using i) Latent Motion Encoding based Inference and ii) Invernet Method

- Chapter 5: Experimental Evaluation - This chapter presents in detail the datasets used, the experimental steps and the evaluation of the different inference models for different embedding models

- Chapter 6: Conclusion and Future Direction - We conclude the thesis with some directions towards future research from our work

Chapter 2

Problem Formulation

2.1 Problem Statement

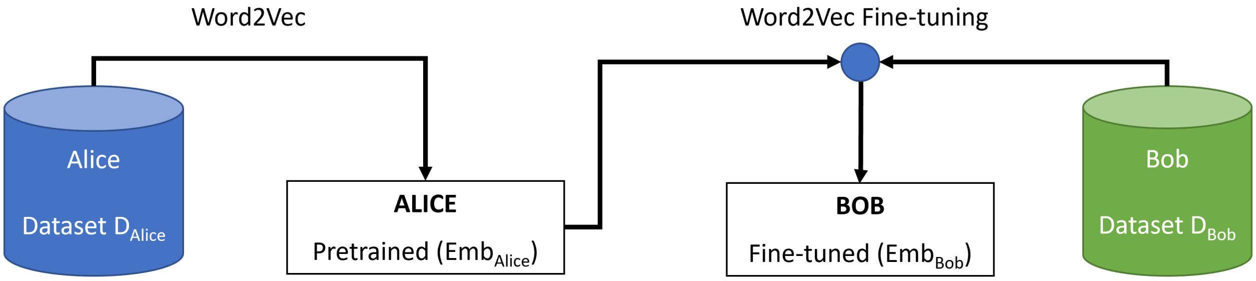


Figure 2.1: Problem statement

The current trend in Natural Language Processing is to fine-tune pretrained models on domain specific downstream datasets. In essence, we can consider the layer weights of the pretrained models as the pretrained embeddings. Traditionally, these pretrained embeddings are used to initialize the weights of the layers for a downstream model. Fine-tuning involves retraining these weights to get better word representations in the downstream domain. Large models like BERT [10] and GPT-3 [8] that are pretrained on huge corpora are often published by organizations with training capacity. Eventually, other organizations or individuals retrain these models on their domain specific corpus to get fine-tuned models whose weights can be considered as fine-tuned word embeddings. With the surge of niche datasets, the NLP community is observing a culture of sharing the fine-tuned models for further fine-tuning. An example of this paradigm can be found in the "Hugging Face Community Model Hub" [41].

In figure 2.1, Alice decides to train word embeddings based on her large dataset $\mathbf{D}_{\text{Alice}}$. She publishes her pretrained word embeddings Φ_{Alice}. Bob fine-tunes Φ_{Alice} on his smaller dataset $\mathbf{D}_{\text{Bob}}$. Bob publicly publishes his fine-tuned word embeddings Φ_{Bob} so that others can further fine-tune Φ_{Bob}. $\mathbf{V}_{\text{Alice}}$ and $\mathbf{V}_{\text{Bob}}$ are sets of all the words from $\mathbf{D}_{\text{Alice}}$ and $\mathbf{D}_{\text{Bob}}$ respectively, excluding the stop-words.

2.1.1 Example

Table 2.1: An example of D_{Bob}

Sentence 1	Condition of economy is good
Sentence 2	Economy is growing

Table 2.2: D_{Bob} after pre-processing and removing stop words

Sentence 1	["condition", "economy", "good"]
Sentence 2	["economy", "growing"]

Table 2.3: Binary context of the word "economy" in $\mathbf{D}_{\text{Bob}}$ within a window size $W_C = 1$

Vocab	condition	economy	good	growing
Sentence 1	1	0	1	0
Sentence 2	0	0	0	1

2.1.2 Assumptions

Assumptions for our proposed breach of privacy are as follows:

- Alice has access to $\mathbf{D}^{\text{Alice}}$ and Φ^{Alice}
- Alice is able to compute a binary word-to-word co-occurrence $\mathbf{C}^{\text{Alice}}$ of her dataset $\mathbf{D}^{\text{Alice}}$
- Alice also gets access to Φ^{Bob} since Φ^{Bob} is published in the public domain
- $\mathbf{V}^{\text{Bob}} \cap \mathbf{V}^{\text{Alice}} \neq \emptyset$

2.1.3 Problem Formulation

Finding out the set of words that constitute Bob's fine-tuned embedding model is trivial because Bob's embedding model is an association between Bob's vocabulary and the corresponding vectors for the words in Bob's vocabulary. So, the question we are asking is: "Can Alice reveal the binary word-to-word co-occurrence $\mathbf{C}^{\text{Bob}}$ from Bob's fine-tuned word embeddings Φ^{Bob} with the knowledge of $\mathbf{D}^{\text{Alice}}$, $\mathbf{C}^{\text{Alice}}$, and Φ^{Alice}?"

We formulate our question as a learning problem where Alice learns a set of mappings $\mathbf{F}$ such that,

$$\begin{aligned} \mathbf{F} = \{\ & f^{\text{word}} \mid \forall \text{ word} \in \mathbf{V}^{\text{Alice}} , \\ & f^{\text{word}} : < \Phi^{\text{Alice}}, \Phi^{\text{Bob}} > \rightarrow\ \mathbf{C}^{\text{Bob}}(\text{word})\ \} \end{aligned} \tag{2.1}$$

When analyzing context leakage, we include only the set of vocabulary $\mathbf{V}$ such that,

$$\mathbf{V} = \mathbf{V}^{\text{Alice}} \cap \mathbf{V}^{\text{Bob}} \tag{2.2}$$

As the attacker, Alice will not have any pretrained embeddings of the words that are not present in $\mathbf{V}_{\text{Alice}}$. So, we exclude such words that only belong to $\mathbf{V}^{\text{Bob}}$ but not $\mathbf{V}^{\text{Alice}}$. Tables 2.1, and 2.2 show an example of $\mathbf{D}^{\text{Bob}}$. In table 2.3, we can see an example of the binary context output of the word "economy" in $\mathbf{D}^{\text{Bob}}$. Within a window size of 1, in sentence 1 of $\mathbf{D}^{\text{Bob}}$ the output for "economy" is [1, 0, 1, 0] which means that the words "condition", and "good" occur as either the immediately previous or next word of "economy".

Table 2.4: Table of Notations

Notation	Definition
$\mathbf{D}^{\text{Alice}}$, $\mathbf{D}^{\text{Bob}}$	Dataset belonging to Alice and Bob respectively
$\mathbf{V}^{\text{Alice}}$, $\mathbf{V}^{\text{Bob}}$	Vocabulary of Alice and Bob's datasets respectively
$\mathbf{C}^{\text{Alice}}$, $\mathbf{C}^{\text{Bob}}$	Binary word-to-word co-occurrence of Alice and Bob's datasets respectively
$\mathbf{C}(\text{word})$	Binary word-to-word co-occurrence between a specific word and the rest of the vocabulary of a specific dataset
$\mathbf{D}^{\text{Train}}$	Training dataset that will be split into pretraining and inference dataset
$\mathbf{D}^{\text{Test}}$	Testing dataset
$\mathbf{D}^{\text{pretrained}}$	Dataset for pretraining word embeddings
$\mathbf{D}^{\text{fine-tuned}}$	Dataset used for creating multiple inference sub-samples
n	Number of inference samples
b	Number of documents per inference sample
$\Phi^{\text{pretrained}}$	Word embeddings pretrained on $\mathbf{D}^{\text{pretrained}}$
$\Phi^{\text{fine-tuned}}$	Fine-tuned word embeddings
θ	Word embedding model weight vector
$\theta_{\text{i}}^{\varepsilon}(\text{word})$	Embedding model weight at i$^{\text{th}}$-dimension for specific word after ε epochs

Chapter 3

Background

3.1 Neural Representation Learning

Textual data is discrete and diverse in nature. Neural network models are unable to work with textual data in its natural form since neural networks are numerical function approximators that can only take numbers as input. Therefore, representing text as numbers is a necessary step when learning neural language models. Over the years, numerous models have been introduced for learning numerical representations of natural language tokens. Such numerical rendition of natural language tokens is now known as token embedding or token vector.

Tokens in natural language are units of data that can vary greatly in syntax and semantics. Examples of tokens can be words, subwords, phrases, sentences or entire documents [4]. If we consider words, there are many similarities among the different words in the context of traditional lexicography and computational linguistics [23]. Similar affinity can be observed within the other tokens of natural language. Various neural network models are able to implicitly capture such similarities through clusters of dense vector representations.

One of the earlier canonical works on learning word vectors is discussed in [34], where the authors have used back-propagation technique to learn word representations. Following work in [6] focuses on learning a statistical model of the word sequence distribution at scale with neural networks. The authors report that such distributed representation of words overcome the curse of dimensionality otherwise found in the naive computation of joint probability function of word sequences. At the same time, such distributed neural representations of words show considerable improvements over n-gram models and work well in longer contexts of textual data [6].

In word embeddings, words are represented as continuous and real-valued dense vectors. So, words are essentially distributed in a vector space of real numbers. The functional aspect of word embeddings is that within the vector space, semantically similar words are clustered together and form a linearly analogical relationship with words from different clusters. As a result, word embeddings have brought breakthroughs in various deep learning-based NLP tasks. However, the success of various deep learning models using word embeddings largely depend on the size of the dataset. The larger the dataset the better the performance. For instance, GPT-3 [8] was trained on almost 500 billion tokens , Google's pretrained Word2Vec model [21] was trained on roughly 100 billion words. Training such large language models demands expensive resources. So, to make these language models more accessible to everyone, large companies train the models on large datasets and share the pretrained embeddings so that others can use transfer learning mechanism to make use of the learned knowledge of these models [33]. Eventually, the pretrained transferred representations are fine-tuned on downstream application specific datasets.

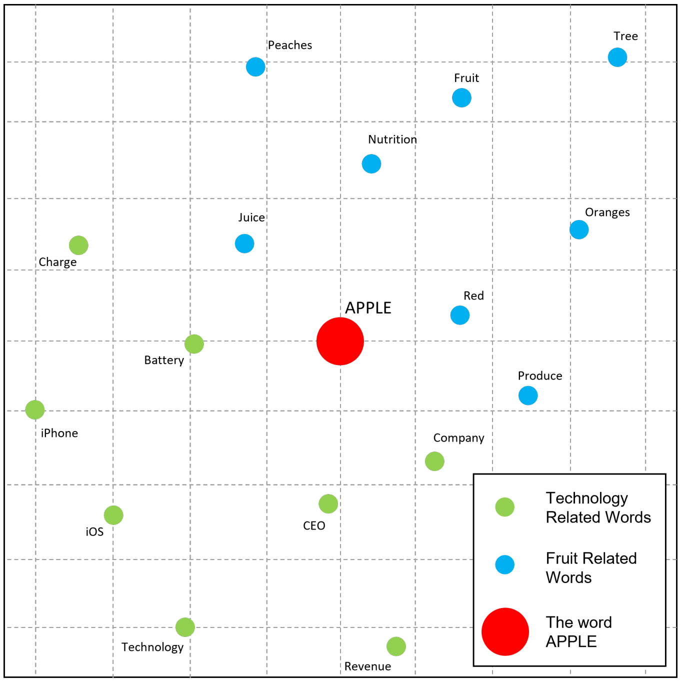


Figure 3.1: Two dimensional t-SNE approximation of d-dimensional word vectors pretrained on a dataset containing both technology and fruits related words

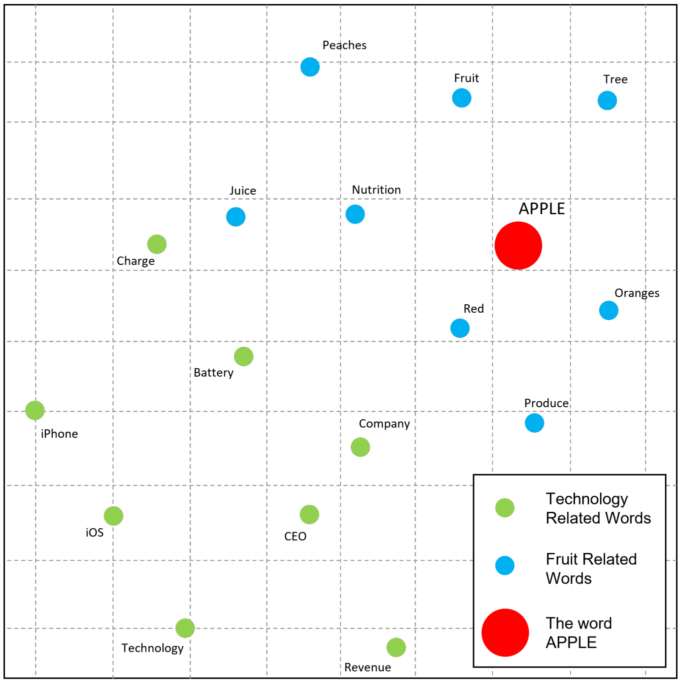


Figure 3.2: Two dimensional t-SNE approximation of d-dimensional word vectors fine-tuned on a dataset containing fruits related words

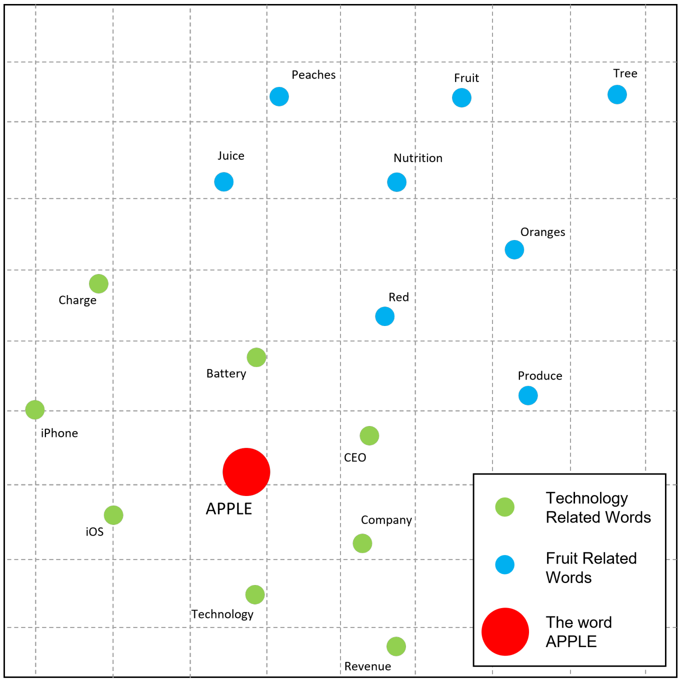


Figure 3.3: Two dimensional t-SNE approximation of d-dimensional word vectors fine-tuned on a dataset containing technology related words

In Fig. 3.1, we can see a two dimensional t-SNE approximation of different word embeddings.

These words belong to either the technology or fruits domain. The word vector for the word "Apple" can be seen positioned on the boundary between the technology and fruits clusters because the "Apple" word vector is equally influenced by both domains. However, in Fig. 3.2 and 3.3, the "Apple" word vector can be seen moving closer to the fruits and technology clusters respectively because the pretrained embedding is fine-tuned on fruits and technology datasets respectively.

Over the years, more than 50 types of neural models have been implemented which can learn the embeddings of natural language tokens [4]. Some of the most popular and relevant models are described as follows:

3.1.1 Word2Vec

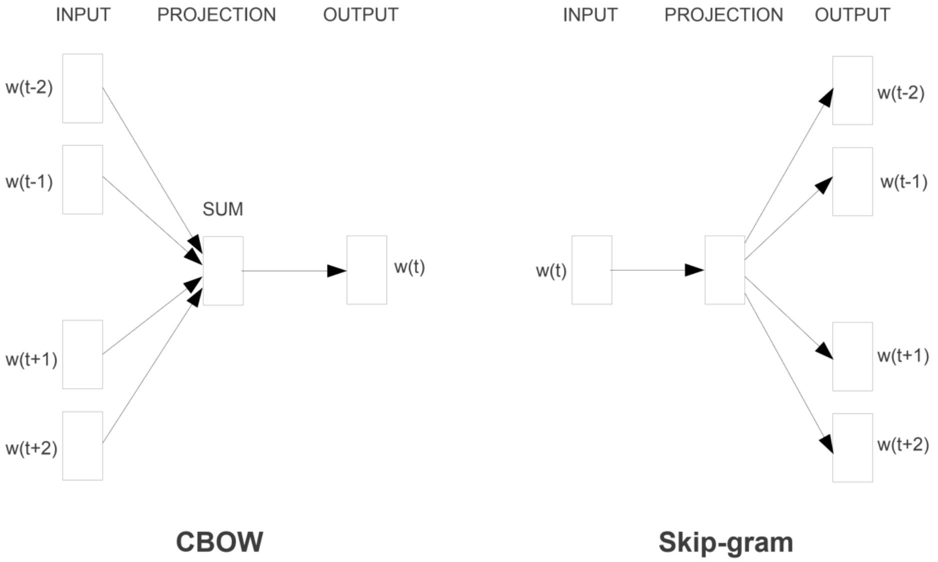


Figure 3.4: Word2Vec models: i) CBOW; ii) SkipGram [20]

In [20], Mikolov et al. introduced two models namely Continuous Bag-of-words (CBOW) and Skip-gram method to learn word embeddings and evaluated the models and the embeddings on a word similarity task. Continuous bag-of-words or CBOW model trains continuous valued word vectors using a simple model. Under the justification that models with non-linear hidden layers tend to be more complex and slower, the CBOW method tries to overcome this complexity by removing the non-linear hidden layer and by using a shared projection layer for the words. The authors have designed the CBOW model [20] as a log-linear classifier where an unknown word is predicted using the four previous and four next words. The resemblance of CBOW to a standard bag-of-words model derives from the fact that CBOW does not take into account the order of words when considering the previous and next words [20]. On the other hand, Skip-gram model trains the word embeddings using an opposite approach compared to the continuous bag-of-words model. Contrary to the CBOW method, the continuous skip-gram method considers a single word and uses a log-linear classifier to predict the neighboring words both before and after the given word. Similar to the CBOW method, the continuous skip-gram model also has a shared continuous projection layer which is integral to computing the continuous vector representation of the words [20].

3.1.2 GloVe

GloVe model [27] also produces continuous word vectors but with a focus on the explainability of the linear regularities of word embeddings. GloVe is trained by combining the best of global matrix factorization and local context window methods [27]. The authors have efficiently utilized the word-word co-occurrence statistics when training their model. Because of considering the global corpus statistics, GloVe is better at reflecting contexts through the produced representations. As a result, the GloVe model has been able to capture semantically sound sub spaces within the word embeddings [27].

3.1.3 BERT

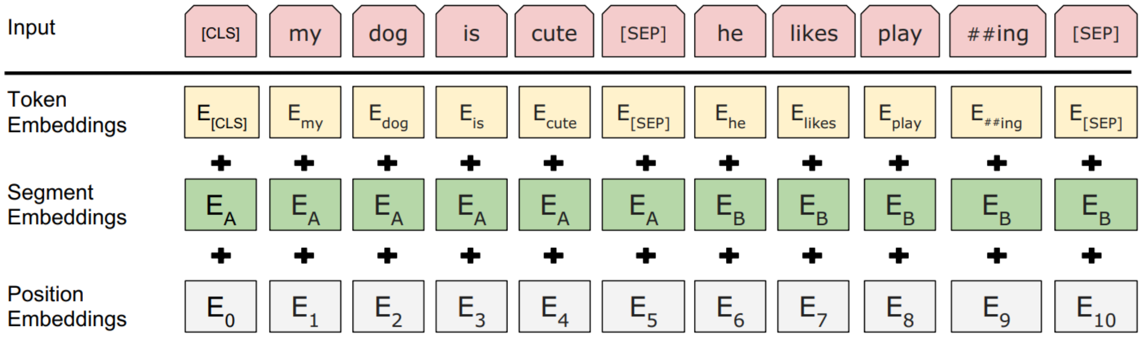


Figure 3.5: BERT input representation [10]

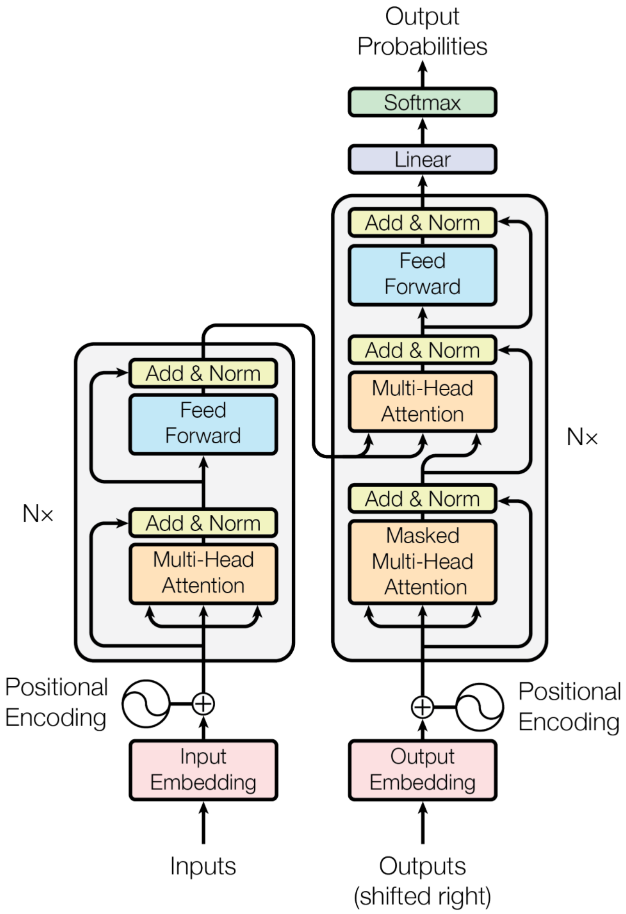


Figure 3.6: Transformer model architecture [39]

Bahdanau et al. introduced an attention mechanism for machine translation task in [5]. The authors claimed that using a variable length encoding of a source sentence using word by word attention for each time step can improve the representational bottleneck of using a fixed-length vector encoding [5]. In [39], Vaswani et al. proposed a simple Transformer architecture that rely solely on attention layers for encoding and decoding instead of expensive recurrent or convolutional operations. Transformers have achieved high performance in machine translation tasks [39]. In an attempt to generalize the use of Transformers, Devlin et al. introduced the BERT model (Bidirectional Encoder Representations from Transformers) [10]. BERT is intended to be used as a pretrained model which uses both left and right context for better performance [10]. Word embeddings trained with the BERT model can be fine-tuned to achieve superior performance in a range of tasks e.g. MultiNLI, Question-Answering etc. [10].

3.2 Transferring Knowledge in Deep Learning

3.2.1 Deep Learning

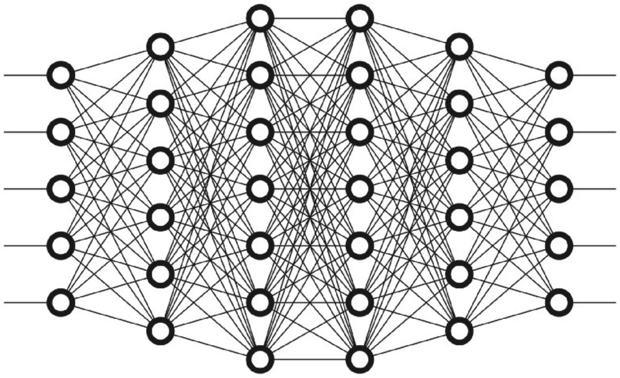


Figure 3.7: An example of a deep neural network [29]

Deep learning is an evolutionary form of Artificial Neural Networks. In Artificial Neural Networks, we try to approximate the mapping between an input and an output using Multi-Layer Perceptrons. Multi-Layer Perceptrons are essentially feedforward networks in which each neuron transforms its input and passes it to the next neuron as an output. Eventually, the error rate between the final output of the network and the ground truth is computed and backpropagated through the network so that the prior neurons can adjust their weight parameters accordingly. Artificial Neural Networks perform well on smaller datasets and with already engineered features. However, for larger datasets, it becomes challenging to handcraft features suitable for the problem. With deep learning, we stack together many such perceptron layers and create an effective hierarchical feature learning environment so manually engineering features is no longer required [14]. An added benefit of deep learning is that with an optimal set of hyperparameters, a deep learning model avoids the plateaue problem and learns to approximate better with more training. Non-linear activation functions in each layer helps the deep neural network to learn non-linear functions as well as linear functions. Moreover, deep neural networks are able to learn better input representations along with a supervised or unsupervised learning objective [14].

3.2.2 Transfer Learning

Transfer Learning is an effective process of transferring learnt knowledge of one model from one domain or task to another model on a different domain or task. It is a shortcut approach to warm start machine learning models and to achieve better downstream task specific performance. Transfer learning is suitable when there is not enough data to train a new model. We can borrow already learnt knowledge in a similar domain and achieve much better performance even if we have a smaller dataset to train our model on. In [24], the authors categorize transfer learning based on what is being transferred. The categories are mentioned below:

i Instance-transfer

ii Feature-representation-transfer

iii Parameter-transfer

iv Relational-knowledge-transfer

A common transfer learning approach in natural language processing is to either transfer models by parameter or to transfer word embeddings in the form of a feature representation transfer. In the case of feature representation transfer, pretrained embedding vectors are used to initialize the downstream model. Then, the downstream model along with the pretrained embedding vectors can be fine-tuned on a downstream corpora with respect to either a supervised task or in an unsupervised way to learn about the semantic and syntactic relationships among words in the downstream dataset [33]. Validation error rate has shown to be very small for a fine-tuned language model as opposed to one trained from scratch [31].

3.3 Information Leakage from Machine Learning Models

3.3.1 Membership Inference

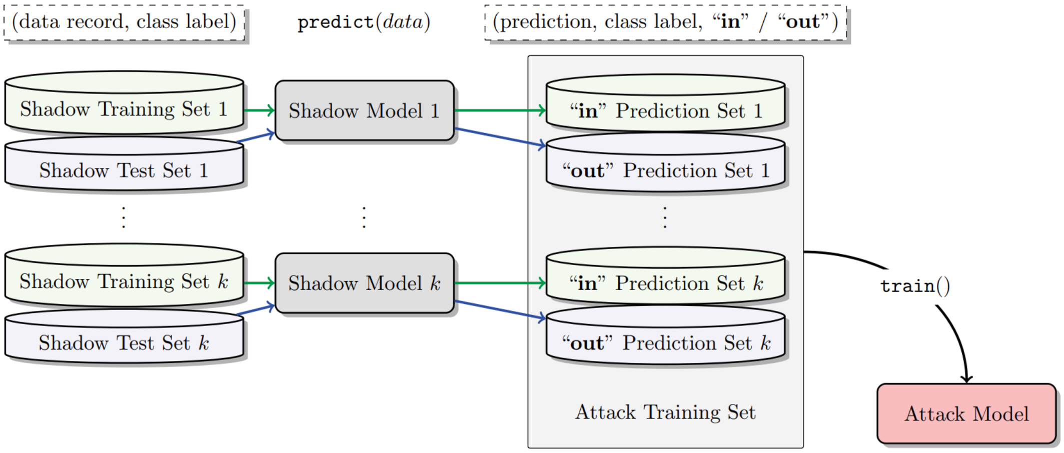
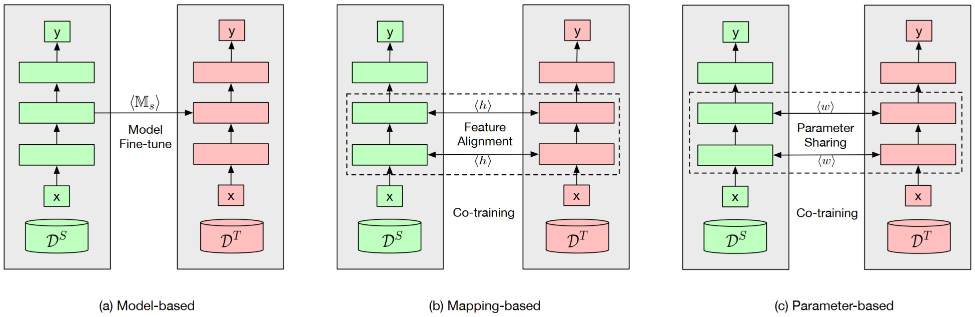


Figure 3.8: Membership inference using shadow model technique [35]

The goal of membership inference attacks is to identify whether a specific piece of data belongs to the training dataset of a target model. Shokri et al. introduced a shadow model technique in [35] for membership inference attacks. The authors assume that the target model will predict a particular target class with a certain probability. The authors select k number of overlapping shadow training sets with some background knowledge about the target dataset [35]. Eventually, they use the k shadow training sets to train k shadow models that predict the target class, the probability of the predicted class and membership within any of the shadow training sets. These outputs are used to train c = | target class | number of attack models that predict whether the provided piece of data belongs to the training set [35].

3.3.2 Information Leakage in Deep Transfer Learning



(a) Model-based

(b) Mapping-based

(c) Parameter-based

Figure 3.9: Leakage in deep transfer learning [9]

Chen et al. demonstrated inference attacks in different transfer learning paradigms such as model based, mapping based, and parameter based [9]. In all these settings, the authors consider that the owner of the downstream target dataset carries out inference attack on the source dataset. The authors applied the shadow model technique from [35] for their model based attack. For the mapping based attack, they aligned the hidden features from both domains across the transfer learning process and used the similar features between both domains to infer properties of the source dataset [9]. The authors assume access to an auxiliary dataset which shares a distributional similarity with the source dataset. For the parameter based method, they train an attack model using the auxiliary dataset to determine batch properties from the transferred gradients [9].

3.3.3 Information Leakage in Embedding Models

In [36], the authors presented an embedding inversion attack on sentence embeddings. In the black-box version of their attack, Song and Raghunathan considered access to a sentence embedding model where querying the embedding model with a sentence gives the vector representation of the sentence [36]. Using an auxiliary dataset to train their attack model, the authors proposed a multi-label classification and a multi-set prediction method to predict which words are present in a sentence given the sentence vector.

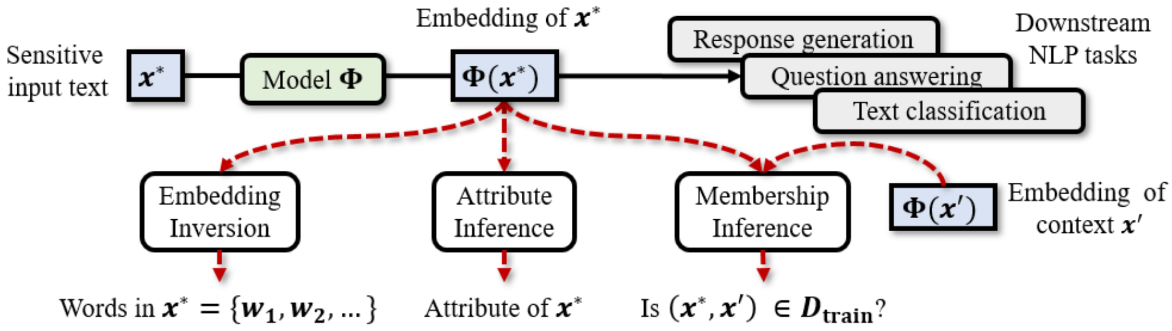


Figure 3.10: Embedding inversion [36]

Chapter 4

Methodology

We develop our methodology to realize the inversion of Word2Vec [20] and GloVe [27] embedding methods. Since we try to infer the binary co-occurrence context from only the pretrained and fine-tuned embeddings, our proposed "Invernet" inversion architecture is model agnostic. In this chapter, we explain in the detail our methods of pretraining, fine-tuning, the input and output representations, and the Invernet framework to invert the embeddings.

4.1 Pretraining

Due to a strong coupling with the nature of the embedding model, the pretraining and fine-tuning steps are tailored individually for the different embedding models. In the pretraining step, we use different embedding models like Word2Vec [20],and GloVe [27] to pre-train word embeddings on the source dataset. Details about how we train the pretrained embeddings using different embedding models are as follows:

4.1.1 Continuous Bag of Words (CBOW)

The goal of the CBOW method is to maximize the log probability of a word given a set of surrounding words. For a window range of [-c, c] around a word w_t at position t, CBOW method optimizes the following function [20]:

$$\log p(w_t|w_{t-c},...,w_{t-1},w_{t+1},...,w_{t+c}) \tag{4.1}$$

4.1.2 SkipGram

On the other hand, the objective of the SkipGram model is to maximize the average log probability of the contextual words given a target word. For a window range of [-c, c] around a word w_t at position t, SkipGram method optimizes the following function [22]:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{4.2}$$

where T is the total number of words in a specific sequence. In Eq. (4.2), $p(w_{t+j}|w_t)$ is fundamentally defined as the following softmax probability [22]:

$$p(w_t|w_c) = \frac{\exp{(v_t^T v_c)}}{\sum_{w=1}^{W}\exp{(v_w^T v_c)}} \tag{4.3}$$

where W is the vocabulary size, w_t and w_c are respectively the center word and the context word, v_t and v_c are respectively the vector representations of the center word and the context word. However, because of computational feasibility we can resort to using either a hierarchical softmax or negative sampling instead of using the basic form of softmax probability in Eq. (4.3) [22].

4.1.3 GloVe

In the case of GloVe, we optimize the following function [27]:

$$\hat{J} = \sum_{i,j} f(X_{ij})(v_i^T\tilde{v_j} - \log X_{ij})^2 \tag{4.4}$$

where Xij is the number of times words i and j co-occur within a certain window, v_i and $\tilde{v_j}$ are the word vectors of words i and j respectively [27]. With an empirical value of $\alpha = 3/4$, $f(X_{ij})$ is a weighting function which in [27] is defined as follows:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & if x < x_{max} \\ 1 & otherwise \end{cases} \tag{4.5}$$

4.1.4 Word Vector Initialization

When training a model from scratch, the word vectors can be initialized to either all zeros or some random values. Sometimes initial values are sampled from special distributions or generated randomly using special hash functions. Word vector initialization plays an important role in the remainder of the training process because properly initialized vectors enable feasible and faster convergence. Without proper initialization, the word embedding model might diverge because of either vanishing gradients or exploding gradients. Even if the model converges without weight initialization, it is possible that the convergence will be slow and the model might become stuck at a local optima. In [19], the authors mention a few different ways of initializing word vectors for pretraining. Some common word vector initialization values of the k^{th} dimension θ_k at epoch 0 are as follows:

$$\theta_k^0 \begin{cases} = 0 \\ = 1 \\ \sim \mathbf{N}(\mu, \sigma^2);\ \text{Normal} \\ \sim \mathbf{N}(0, \sqrt{2/(fan\_in + fan\_out)};\ \text{Glorot Normal [13]} \\ \sim \mathbf{N}(0, \sqrt{2/fan\_in});\ \text{Glorot Uniform [13]} \\ \sim \mathbf{U}(-\sqrt{6/(fan\_in + fan\_out)}, \sqrt{6/(fan\_in + fan\_out)});\ \text{He Normal [18]} \\ \sim \mathbf{U}(-\sqrt{6/fan\_in}, \sqrt{6/fan\_in});\ \text{He Uniform [18]} \end{cases} \tag{4.6}$$

where μ and σ^2 are respectively the mean and the standard deviation of a normal distribution, "fan_in" and “fan_out” are respectively the number of input units and the number of output units to and from the k^{th} neuron. If we pre-train the d-dimensional word vectors for ε number of epochs, the pretrained vector for a specific word becomes,

$$\Phi^{pretrained}(word) = [\theta_0^{\varepsilon}(word), \theta_1^{\varepsilon}(word), ..., \theta_d^{\varepsilon}(word)] \tag{4.7}$$

4.2 Fine-tuning

Fine-tuning is the process of updating pretrained word vectors based on a downstream domain or downstream task or both. For unsupervised fine-tuning, word vector updates are not constrained with respect to a specific downstream task. In a word embedding model, word vectors are essentially a set of model weights. In our study, we consider unsupervised fine-tuning of word vectors using Word2Vec and GloVe models. Our fine-tuning process is not governed by any specific downstream task objective although having a supervised task objective can train even better downstream fine-tuned embeddings, potentially rendering the embeddings even more vulnerable. Instead we focus on an unsupervised setting where the word vectors are initialized using pretrained vectors and then updated in the form of model weights based only on the semantic and syntactic regularities of the downstream corpora.

In the case of pretraining, we initialize the word vectors using any option from Eq. (4.7). However, instead of training the word vectors from scratch with any of the initialization from Eq. (4.7), in fine-tuning, we initialize the word vector for a specific word with the pretrained values as follows:

$$\theta^{\varepsilon+1}(word) = [\Phi_0^{pretrained}(word), \Phi_1^{pretrained}(word), ..., \Phi_d^{pretrained}(word)] \tag{4.8}$$

Then, using gradient descent method and backpropagation technique, we update word vectors or the model weight parameters as follows:

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta J(\theta^{t-1}) \tag{4.9}$$

where $0 < i < n_{epochs}$ and θ^{t_i} is the state of the weight vector at epoch t_i. η is the learning rate hyperparameter which can be manually adjusted to control the stability of the gradient descent. ∇_θ is the gradient or the dimension-wise partial derivative of the cost function $J(\theta)$ with respect to the weight vector $\theta^{t_{i-1}}$.

After fine-tuning the word vectors for p epochs, we get the final fine-tuned word vector representation for a specific word as follows:

$$\Phi^{fine-tuned} = [\theta_0^{\varepsilon+p}, \theta_1^{\varepsilon+p}, ..., \theta_d^{\varepsilon+p}] \tag{4.10}$$

4.3 Invernet Framework

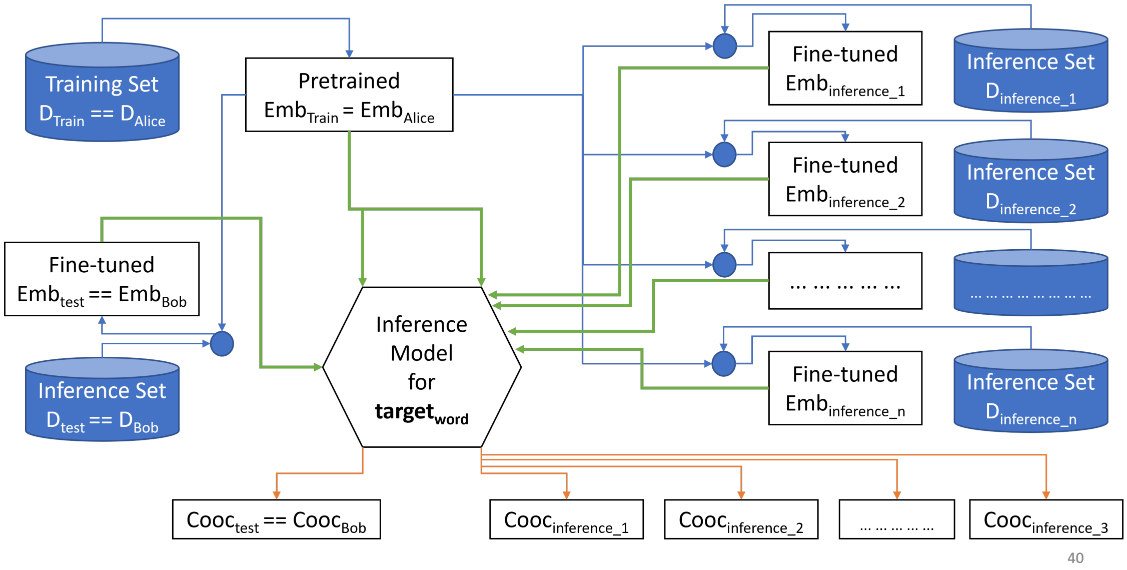

Training Set D_{Train} == D_{Alice}
Pretrained Emb_{Train} = Emb_{Alice}
Fine-tuned Emb_{test} == Emb_{Bob}
Inference Set D_{test} == D_{Bob}
Inference Model for **target**$_{word}$
Fine-tuned $Emb_{inference\_1}$
Inference Set $D_{inference\_1}$
Fine-tuned $Emb_{inference\_2}$
Inference Set $D_{inference\_2}$
… … … … …
… … … … … … … …
Fine-tuned $Emb_{inference\_n}$
Inference Set $D_{inference\_n}$
$Cooc_{test}$ == $Cooc_{Bob}$
$Cooc_{inference\_1}$
$Cooc_{inference\_2}$
… … … … …
$Cooc_{inference\_3}$
40

Figure 4.1: Inversion model framework ("Invernet")

The objective of the Invernet inversion framework is to invert fine-tuned word embeddings to the binary context distribution of a specific target word. Our proposed Invernet framework contains multiple components namely a focused document selector, an inference data sampler, input embeddings processor, and finally an inference model.

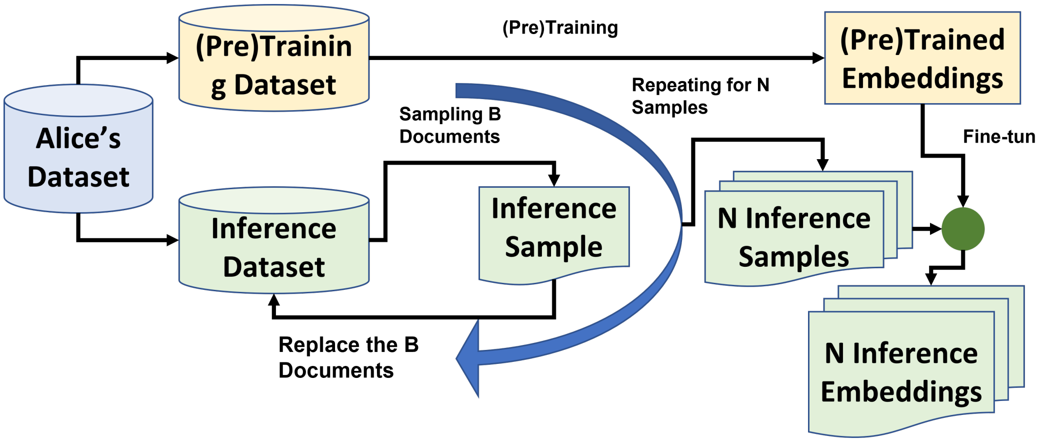


Figure 4.2: Inference Sampler

At first, from the dataset vocabulary we select a set of target words for which we wish to inference the contextual information from the downstream dataset. We can either select the target words based on information need or based on random uniform sampling. Then, for each target word we carry out a focused selection method to select a subset of documents containing the target word, from the dataset. We split the dataset into source, inference, and testing dataset. We pre-train a set of word embeddings in an unsupervised setting using the source dataset and different embedding models. Then as depicted in Fig. 4.2, using a random uniform sampling approach we select a set of **B** articles, **N** number of times from the inference dataset to create a pool of inference sub-datasets. Here, **B** represents the bin size and **N** represents the number of inference samples being sampled. Then, we fine-tune those **N** number of inference samples and produce **N** number of inference embeddings. Then, using the pretrained embedding and all the **N** number of inference embeddings, we train our inference model so that it can learn the relation between word embeddings and the corresponding target word context. The algorithm of the Invernet framework is given below:

Algorithm 1: Invernet Algorithm

```
Input: $\mathbf{D}^{\text{Train}}$, $\mathbf{D}^{\text{Test}}$, target_word, n, b
$\mathbf{D}^{\text{pretrained}}, \mathbf{D}^{\text{inference}} \leftarrow$ split_dataset($\mathbf{D}^{\text{Train}}$)
$\Phi^{\text{pretrained}} \leftarrow$ pre-train embeddings on $\mathbf{D}^{\text{pretrained}}$
for $i \leftarrow 0$ to $n$ do
    data$^{\text{i}} \leftarrow$ uniformly & randomly sampled b documents from $\mathbf{D}^{\text{inference}}$
    $\Phi^{\text{i}} \leftarrow$ fine-tune $\Phi^{\text{pretrained}}$ on data$^{\text{i}}$
    $\Phi^{\text{fine-tuned}} \leftarrow$ concatenate $\Phi^{\text{fine-tuned}}$ & $\Phi^{\text{i}}$
    $\text{C}^{\text{i}} \leftarrow$ binary co-occurrence vector between target_word and all vocabulary of data$^{\text{i}}$
    $\text{C}^{\text{fine-tuned}} \leftarrow$ concatenate $\text{C}^{\text{fine-tuned}}$ & $\text{C}^{\text{i}}$
Train an inference model $f$ that predicts $\text{C}^{\text{fine-tuned}}$ from $< \Phi^{\text{pretrained}}$, $\text{self}_{\text{attn}}(\Phi^{\text{pretrained}}, \Phi^{\text{pretrained}}, \Phi^{\text{pretrained}})$, $\text{self}_{\text{attn}}(\Phi^{\text{fine-tuned}}, \Phi^{\text{fine-tuned}}, \Phi^{\text{fine-tuned}})$, $\Phi^{\text{fine-tuned}} >$
```

4.3.1 Preparing Embeddings for Inference

We prepare a two dimensional co occurrence matrix from the embedding dictionaries where each row of the matrix is a vector for a specific word. We keep track of the word index of the embedding matrix through a separate map between words and indices. We maintain one set of word embedding matrix for the pretrained embeddings. After fine-tuning, we find the word intersection between the source and the downstream model and create another set of word embedding matrix which accounts for the changes effected on the pretrained embeddings by the fine-tuning process.

We apply self-attention technique on both the pretrained and fine-tuned word vector to later pass as input to the inference model. The motivation for using self-attention individually on the pretrained and fine-tuned embeddings is to find the degree of influence among different word vectors within the pretrained and fine-tuned embeddings separately. Since the influence among words within a specific set of embeddings entirely depend on their relative positions in the training corpus, we have found the self-attention method very useful in identifying the degree of interaction among words within a specific training dataset. We use the following formula [39] to calculate the self-attention scores of each of the pretrained and fine-tuned target word vectors:

$$self\_attn(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4.11}$$

where Q, K, and V are respectively the query, key, and value vectors. d_k is the number dimension size of the query and the key vectors [39].

Eventually, we use the concatenation of the source vector of a target word, the self-attention scores of the source vector, the self-attention scores of the fine-tuned vector, and the fine-tuned target word vector as input to the inference model. The input can be defined as follows:

$$v_s = \Phi^{pretrained}(target\_word) \tag{4.12}$$

$$v_f = \Phi^{fine-tuned}(target\_word) \tag{4.13}$$

$$x =< v_s, self\_attn(v_s, v_s, v_s), self\_attn(v_f, v_f, v_f), v_f > \tag{4.14}$$

where v_s and v_f are respectively the pretrained and fine-tuned word vectors for the target word, and x is the concatenated input to the inference model.

4.3.2 Inference Model

We use a deep neural network to implement the mapping function in Eq. (2.1). We stack together a series of dense layers with a combination of dropout layers [37] as the inference model. The dense layers learn the weights of the network and the dropout layers prevent overfitting. We provide a concatenation of the pretrained embeddings and the fine-tuned embeddings as the input to the inference model. So, the model has two input layers followed by a concatenation layer that prepares the final input for the model. Finally, we have an output layer that predicts the sigmoid probability of whether a word appears in the binary context of the target word. The output of the n$^{\text{th}}$-layered model can be defined as follows:

$$z_j^{[n]} = \sum_k w_{jk}^{[n]} a_k^{[n-1]} + b_j^{[n]} \tag{4.15}$$

$$a_j^{[n]} = g^{[n]}(z_j^{[n]}) \tag{4.16}$$

where z_j is the non-activated output for a certain neuron i.e. j^{th} neuron of the n^{th} layer, $w_{jk}^{[n]}$ is the weight of the synapse between the j^{th} neuron of the n^{th} layer and the k^{th} neuron of the $(n-1)^{st}$ layer, $a_k^{[n-1]}$ is the activated output of the k^{th} neuron from the $(n-1)^{st}$ layer, $b_j^{[n]}$ is the bias of the j^{th} neuron from the n^{th} layer, $g^{[n]}$ is the non-linear activation function of the n^{th} layer.

In case of the hidden layers, we use Rectified Linear Unit (*ReLU*) as the activation function. So, Eq. (4.16) becomes,

$$a_j^{[n]} = ReLU(z_j^{[n]}) = max(0, z_j^{[n]}) \tag{4.17}$$

For the output layer, we use sigmoid as the activation function. So, with a single neuron in the last layer, Eq. (4.16) becomes,

$$a^{[n]} = sigmoid(z^{[n]}) = \frac{1}{1+e^{-z^{[n]}}} \tag{4.18}$$

4.3.3 Inference Context Distribution

We have chosen the output of the neural network to be the binary context vector of a target word. So, we have to repeat the training multiple times i.e. once for each word. An example of our output vector is shown in Table 2.3. The output of the model is a sigmoid probability between 0 and 1. We apply a set of thresholds over which the output becomes 1 and otherwise the output becomes 0. If the output at a specific index becomes 1, it means that the word at that index is present in the context window of the target word. From a design perspective, an alternative option would be to use the complete co-occurrence matrix as the output. However, using the complete co-occurrence matrix as output entails a multi-output classification which tends to be challenging to train [43].

4.3.4 Loss Function

During the training process, we have used a Cosine Similarity based loss function where we try to minimize the cosine distance between the prediction and the ground truth. The domain S and the loss function *L* can be formulated as follows:

$$S = \{(x_i, y_i)\}_{i=1}^{n} \tag{4.19}$$

$$L_S(\theta) = -\sum_{i=1}^{n}(||f_\theta(x_i)||_2 \times ||y_i||_2) \tag{4.20}$$

Chapter 5

Experiments

In this section we describe the experimental details regarding the dataset, preprocessing, and the training process. Then we evaluate the performance of our inversion model on different Word Embedding models.

5.1 Datasets

We have used two datasets for training and testing the embedding inversion model. The first dataset is the AG News dataset [16] [45]. AG News dataset contains 4 classes spanning 120,000 sample articles. After preprocessing, we get ∼60000 unique words in the dataset. The second dataset is a collection of New York Times (NYT) articles [44]. The NYT dataset contains news articles on 59 different sections (e.g. Sports, World etc.). We use all 6013 articles from the *World* section of the NYT dataset to conduct our experiments. After removing the stopwords, the vocabulary size of the *NYT World* news articles reach 5633.

5.2 Training Details

5.2.1 Preprocessing

As part of the preprocessing, we clean both the datasets by removing any special character. Then we replace all the numbers in the datasets by a special "<num>" tag. We perform case-folding on all the words in the datasets. We remove all the stop words from the datasets using the NLTK English Stop Words Corpus [7] as reference. Then, we tokenize all the space delimited articles in the datasets so that each entry of the dataset is a list of words.

5.2.2 Focused Data Selector

We follow a focused selection approach where we consider only the subset of documents from the dataset that contain the target word. Our motivation for adopting a focused selection approach arose from the fact that we are using word-to-word co-occurrence values for training our inference model. So, if the target word does not appear in an article, we cannot establish with confidence the relation of the words in that article with the target word. Benefits of using this focused selection approach instead of a global selection approach are two folds. The benefits are as follows:

- Considering the words that do not occur with the target word within the same article as noise and removing those noisy words for a more focused inversion
- Saving computational time by removing the noise and reducing the dataset

We can define the dataset that is selected with focus on the target word as follows:

$$\mathbf{D}^{target\_word} = \{d | \forall d \in D^{original}, target\_word \in d\} \tag{5.1}$$

$$\mathbf{D}^{target\_word} \subseteq D^{original} \tag{5.2}$$

where D_{target_word} is the focused dataset from the original dataset $D_{original}$ with respect to the target word.

5.2.3 Splitting Dataset

We have different news categories in both of the datasets. For each set of experiment, we pick one specific news section and hold out around 10% documents from that section as the test set. Then, we pick around 1000 documents from the remainder of that section and combine it with 1000 documents from each of the other news sections to form the training data. In this way, we are ensuring that the distribution of the test is not exactly the same as the training set. Eventually, we split the training set into 50% split for pretraining word embeddings and 50% split as the inference dataset.

5.2.4 Pretraining

We perform pretraining in different ways for different embedding models namely Word2Vec (CBOW and SkipGram method) [20], and GloVe [27]. For both Word2Vec based CBOW and SkipGram models, we pre-train using the widely used Gensim python module [30]. In the case of GloVe [27], we have used the original source code provided by the authors in [1] to pre-train the embeddings on our dataset. In all the pretraining algorithms, we use a vector size of 20 as for our dataset size, we observed well structured word clusters with 20 dimensional word vectors. In [25], the authors have mentioned that starting from a lower bound on dimension size, word embeddings perform acceptably well. The authors have found 19 as the lower bound on word embedding dimensions for their dataset [25]. Another reason to use a lower bound on vector dimension was to reduce overfitting of the inference model since the inference model takes the word vectors as input. We use a minimum word count of 3 for a word to appear in the embedding set for all the embedding models. For GloVe, we use a window size of 5 to determine the co-occurrence statistics.

5.2.5 Fine-tuning

We also adopt three separate methods for fine-tuning the pretrained embeddings. For embeddings trained with Word2Vec, we fine-tune using the Gensim [30] package. For GloVe embeddings, we use Mittens [11] package since Mittens allows a fast vectorized implementation of GloVe that also has provisions for retrofitting.

5.3 Baselines

We use four different experimental approaches for the inference model. In our first approach, we implement a naive baseline using logistic regression model without any inference sub-sampling. In our second approach, we experiment with a motion based inference baseline. In our third approach, we use an ensemble of inference models in the form of neural networks and apply a stacked generalization technique to aggregate their results. For the fourth approach, we generate multiple sets of word embeddings and corresponding co-occurrence statistics from sub-sampling the dataset and then apply a single neural inference model to learn the mapping between the embeddings and the co-occurrence statistics. We call this fourth approach the Invernet inversion model. For all the approaches, we repeat the process once for each target word. A collection of target words can be formed using uniform and random sampling from the entire vocabulary of the pretraining dataset or the collection of target words can also come from the information need or the domain expertise of the attacker. For each target word, we use the same pretrained word embeddings for all the different approaches. We have used Scikit-learn library [26] for the logistic regression based naive baseline and Tensorflow-Keras library [2] for the other three approaches. Below we discuss about our experimental setup for each of the approaches:

5.3.1 Naive Baseline

In this naive approach, we do not perform any inference data sub-sampling. Instead, we fine-tune on the entire inference dataset. We train a logistic regression model to learn the mapping from this fine-tuned embedding and the binary context vector for the target word. Then we use the trained inference model on the test dataset and evaluate the performance of the logistic regression based naive baseline without any inference sub-sampling.

5.3.2 Motion based Inference Model

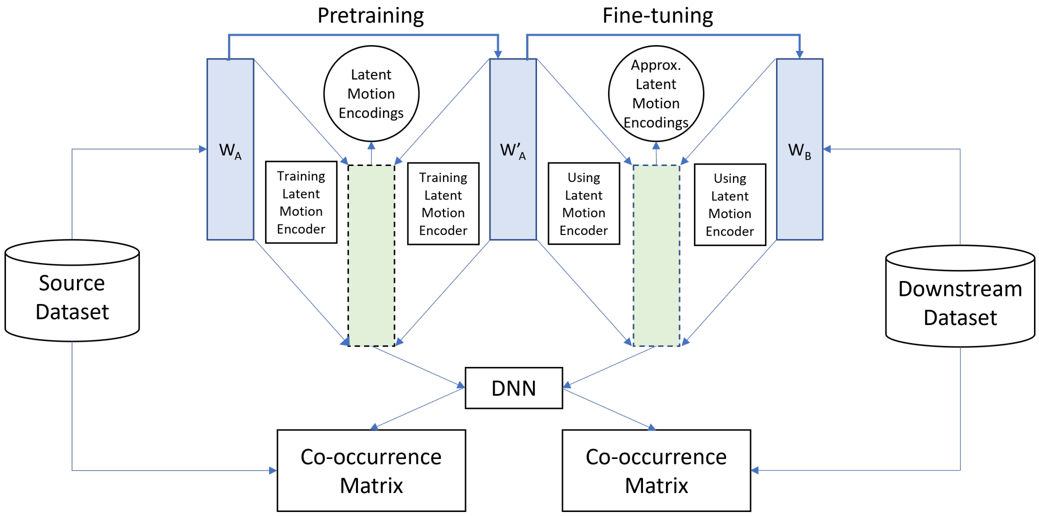

Pretraining
Fine-tuning
Latent Motion Encodings
Approx. Latent Motion Encodings
W_A
W'_A
W_B
Training Latent Motion Encoder
Training Latent Motion Encoder
Using Latent Motion Encoder
Using Latent Motion Encoder
Source Dataset
Downstream Dataset
DNN
Co-occurrence Matrix
Co-occurrence Matrix

Figure 5.1: Motion based inference method

In the motion based inference model, we utilize the various movements of the word vectors during their training process to learn about their context. Word vectors are simply points in a high dimensional space. These vectors are influenced by the other vectors in the same space depending on the proximity of other words with a specific word in the corpus. With respect to the premises of the studied attack, the attacker has access to both the pretraining dataset and the pretrained word embeddings. The attacker is also able to oversee the embeddings training process. In the motion based inference model, we use the epoch by epoch positions of the word vectors and trains a latent motion encoder that takes as input the initial representation of the pretrained embeddings and the final representation of the pretrained embeddings and outputs a latent representation of their motion during training. We have used a series of convolutional lstm layers in the latent motion encoder in order to capture the spatio-temporal changes of the word vectors. Eventually, the motion based inference model learns a mapping between this latent motion encoding and the binary context of the target word in the pretraining dataset.

Since the attacker is not able to oversee the downstream fine-tuning process, the attacker will not have access to the epoch by epoch positions of the word vectors during the fine-tuning process. However, with regards to the premises of the attack, the attacker has access to the initial and the final representations of the fine-tuned word vectors. The initial representation of the word vectors is essentially the final representation of the pretrained word vectors. Therefore, without the epoch by epoch positions of the fine-tuned word vectors, we can only use the pretrained latent motion encoder to approximate the latent motion encoding of the fine-tuned word vectors. Then, using the approximate latent motion encoding of the fine-tuned word vectors, we can use the pre-trained motion based inference model to predict the binary context of the target word in the private downstream dataset.

5.3.3 Stacked Generalization

In this approach, at first we fine-tune the pretrained word embeddings on the entire inference dataset without any sub-sampling. Then, we train multiple inference models to learn mappings between the fine-tuned word embeddings and the binary context vector of a specific word in the inference dataset. The motivation to train multiple models was to have each model learn different patterns from the fine-tuned embeddings that could finally lead to the prediction of which words co-occur with a specific target word. Finally, We use a stacked generalization technique [42] to aggregate the results using a single generalizer. Essentially, in this setup we are creating a committee of inference models. We train the committee members individually. During testing, we take the opinion of all the committee member models and let the generalizer decide on the final binary co-occurrence context of the target word. For all the neural layers in the ensemble and the generalizer, we use *he normal* [18] initializer and *ReLU* activation function. As can be seen from the Fig. 5.2, we have used a combination of 2-dimensional convolutional layers with fully connected dense layers to build the ensemble models. For the generalizer, we have used a series of fully connected dense layers.

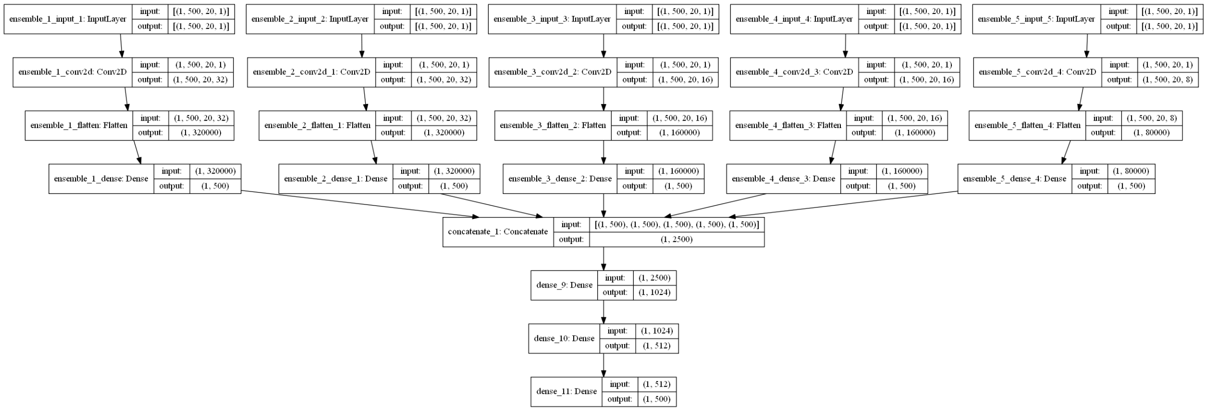


Figure 5.2: Training graph for stacked generalization setting

5.3.4 Invernet Inference Model

From the inference dataset, we sub-sample N (number of inference samples) number of sub-datasets each with a randomly and uniformly sampled B (inference sample bin size) number of articles from the inference dataset. After each sub-sampling step, we replace the articles back into the inference dataset pool so that they might be sampled again by random chance. We experiment with $N = 5, 15, 30$ and $B = 5, 25, 50$. E.g. if $N = 5$ and $B = 5$, we randomly take $B = 5$ articles from the inference dataset and create an inference sample. Then, we return those $B = 5$ articles to the inference dataset pool and randomly sample $B = 5$ articles from the inference dataset again. In this way, we create $N = 5$ inference samples each with $B = 5$ articles. We compute $N = 5$ binary co-occurrence context vectors of the target word from the N inference samples. These binary co-occurrence context vectors are later treated as the output of the inference model. Then we fine-tune the pretrained word embeddings on the inference samples and generate N number of fine-tuned word embeddings. In order to prepare the input, we concatenate the pretrained target word vector, the self-attention scores of the pretrained and the fine-tuned target word vector, and the fine-tuned target word vector itself.

The motivation for adopting such a sub-sampling approach is rooted in the core principles of Machine Learning where we learn patterns from the data. In the Invernet framework, we consider the inference samples as the data and train the inference model based on features extracted from the data in the form of word vectors and the self-attention scores of the word vectors. We ask the following question: "If the word vectors change in a certain way between the pretrained and the fine-tuned embeddings, what binary co-occurrence context of the target word might have induced the said change?" Hypothetically, the more the number of samples, the better the performance of the inference model.

We vertically stack all the *N* number of fine-tuned word embeddings, *N* repetitions of the pretrained embeddings and the corresponding self-attention scores. Then, we concatenate these together and create a single input for the single neural inference model. Again, we use *he normal* [18] initializer and *ReLU* activation function. The initializer helps to constrain the initial weight values within a certain range so that we can avoid vanishing and exploding gradients from faulty initialization. We use a *cosine similarity* based loss function and *Stochastic Gradient Descent* optimizer with a 0.01 learning rate. After experimenting with binary cross-entropy loss, cosine similarity based loss, and a contrastive loss function, empirically we have achieved the best results with cosine similarity based loss function. The training graph of the Invernet inference model is given below:

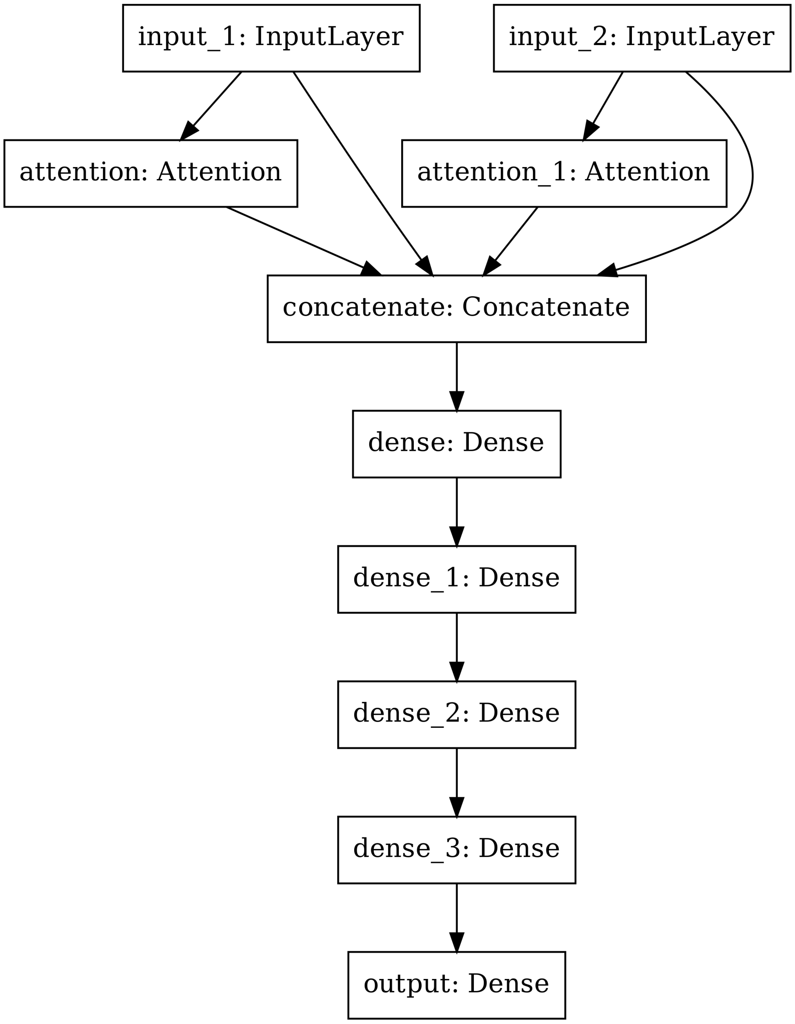


Figure 5.3: Training graph for Invernet inference model

5.4 Classification Report

The goal of our proposed inversion framework is to meet the information need of the downstream dataset from an attacking point of view. In 4.3.3, we have defined the output of our inference model as the inference context distribution of the downstream dataset with respect to a target word. So, we consider our model successful if it is able to infer which words co-occur with a target word within a specific window in the downstream dataset. The output of our inference model is a vector whose elements represent words from the vocabulary. If the value at a specific position of the output vector for a target word becomes 1, then we say that the word at that position is within the context of the target word within a specific window. Now if the word at that is actually present in the target word's context window, then we consider this position of the output vector a true positive (TP). If the word at that position is actually not present in the target word's context window, then we consider this position of the output vector a false positive (FP). However, if the value at a specific position of the output vector for a target word becomes 0, then we say that the word at that position is not present within the context of the target word within a specific window. If the word at that position is actually not present in the target word's context window, then we consider this position of the output vector a true negative (TN). On the other hand, if the word at that position is actually present in the target word's context window, then we consider this position of the output vector a false negative (FN). So for qualitative evaluation purposes our problem is a classification problem.

Although accuracy is a popular classification metric, we do not use accuracy because accuracy can be misleading for datasets with imbalanced class distribution. Since the inference context distribution of a target word can be imbalanced, we use precision and recall to get a better classification performance evaluation. Precision is the fraction of positively classified instances that are actually positive. Recall is the fraction of actual positive instances that are classified as positive. We provide the formula for calculating precision and recall below:

$$precision = \frac{TP}{TP + FP} \tag{5.3}$$

$$recall = \frac{TP}{TP + FN} \tag{5.4}$$

From Eq. (5.3) and Eq. (5.4), we can see that if the inference model classifies a larger portion of the words as positives then the recall might increase but the precision drops. On the other hand, if the inference model identifies correctly the words that are not present within the context window of the target word, the recall increases. However, if the model is unable to predict which words are actually present within the context window of the target word, the precision drops. The model can try to increase the precision by becoming very selective about classifying positive words. But then there is a chance that the recall might drop. So, we can understand that the precision and recall are competing metrics which sometimes can have very polar values. In order to dampen the polar values of precision and recall, we use the F1 evaluation score and Receiver Operating Characteristic (ROC) analysis along with Area Under the ROC Curve (AUC) score.

F1 score is the harmonic mean of precision and recall. Since F1 score combines the precision and recall metrics, F1 score is an excellent and justified metric to compare the classification performance of the different models. We report the F1 scores as representative of the aggregate precision and recall of the different inference models. Formula for the F1 score is as follows:

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{5.5}$$

Receiver Operating Characteristic (ROC) curve is an alternative and a visual way of evaluating different classifier models. ROC is a probability curve between the true positive rate (TPR) and false positive rate (FPR) of a classifier. The TPR is also called the *sensitivity* of a model. Since true negative rate (TNR) is called the specificity of a model, FPR is usually referred to as $(1 - specificity)$ of a model. Formula for TPR, TNR, and FPR are as follows:

$$TPR = \frac{TP}{TP + FN} = sensitivity \tag{5.6}$$

$$TNR = \frac{TN}{TN + FP} = specificity \tag{5.7}$$

$$FPR = \frac{FP}{TN + FP} = 1 - specificity \tag{5.8}$$

In an ROC curve, the x-axis is the FPR or $1 - specificity$ and the y-axis is the TPR or $sensitivity$. An ideal performance of a classifier will be to have a high TPR at a low FPR. So, higher convexity of the ROC curve towards the top left of the ROC space is desirable. Since we are generating individual classification reports for each target word, we will be getting an individual ROC curve for each target word. We are presented with the challenge of averaging the ROC curves of all the target words for a specific inference model. We follow the vertical averaging approach in [12] to average the ROC curves of all the target words for a specific inference model. At first, we consider various sigmoid thresholds for binary classification and note the pair of (TPR, FPR) for each threshold and for each target word. Then we consider only the set of *FPRs* {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0} and find the corresponding highest TPR from all the target word ROC curves. If for a specific target word, none of the (TPR, FPR) pairs have any FPR value from the above set, we interpolate between the available FPR values and find the corresponding TPR for the unavailable FPR. Once we have sufficient pairs of (TPR, FPR), we apply the trapezoidal rule [12] to calculate the area under the ROC curve.

5.4.1 Baseline Comparison

In the following table, we compare the F1 score and the AUC score of the different baseline models and our proposed Invernet framework for the AG News dataset [16][45]. We consider the same 100 unique target words for the different models. We use a random and uniform sampling approach to select the target words sample from the corpus. Then we carry out the inversion process for each of the unique target words and report the average F1 scores and AUC scores of the different models for the target words below:

Table 5.1: Comparing average F1 and AUC scores of different methods on *AG_News* Dataset

Method	Remarks	F1 Score	AUC Score
Motion Inference	Latent Motion Encoding Based Inference Model	0.35	0.48
Stacked Generalization	No inference sub-sampling; Stacked Ensemble of Inference Models	0.50	0.55
Logistic Regression	No inference sub-sampling; Single Inference Model using Logistic Regression	0.58	0.62
Invernet	Inference sub-sampling from inference dataset; Single Inference Model using Deep Neural Network	0.70	0.78

We use a similar approach for the NYT Dataset [44] by randomly and uniformly sampling the same 100 unique words for all the models. We report the average F1 scores and AUC scores of the different models after carrying out the inversion process for all these 100 unique words.

Table 5.2: Comparing average F1 and AUC score of different methods on *NYT* Dataset

Method	Remarks	F1 Score	AUC Score
Motion Inference	Latent Motion Encoding Based Inference Model	0.39	0.51
Stacked Generalization	No inference sub-sampling; Stacked Ensemble of Inference Models	0.52	0.57
Logistic Regression	No inference sub-sampling; Single Inference Model using Logistic Regression	0.60	0.59
Invernet	Inference sub-sampling from inference dataset; Single Inference Model using Deep Neural Network	0.71	0.80

The motion inference method uses an approximation of the latent motion encoding of the downstream dataset. Because we are only approximating the motion encoding and not exactly inferring it for the downstream dataset, we have the worst performance from the motion inference model. The stacked generalization method uses different inference models on the same inference dataset in hopes of deriving different patterns of data from the different models. However, due to the complexity of the models and the method, the stacked generalization method suffers from poor generalization and achieves poor performance. The logistic regression model is a straightforward simple model that too infers binary context from the same inference dataset. But, it has better generalization ability and performs slightly better. Invernet demonstrates the best performance among all these models because of multiple inference sub-sampling and a straightforward deep neural network.

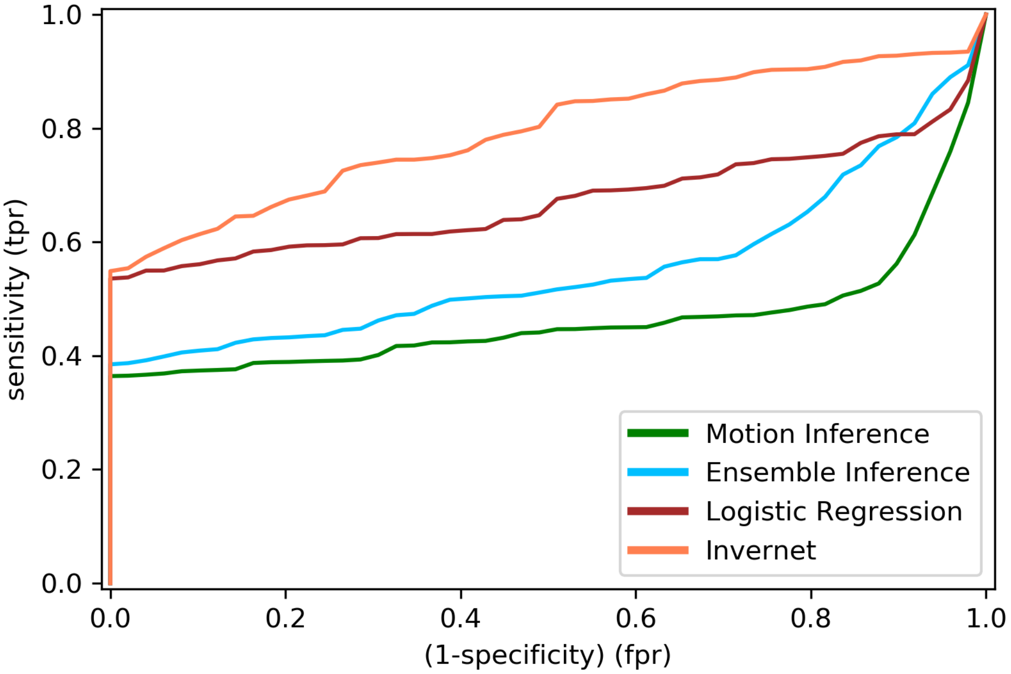


Figure 5.4: ROC analysis for different baselines

5.4.2 Ablation Study

We have performed an ablation study by considering the number of inference samples N and the bin size B as hyperparameters. We use a set of values {5, 15, 30} for N and {5, 25, 50} for B. For each value of N, we consider all the bin sizes from B. Here, number of inference samples N refers to the number of article collections we consider for inference model training. B refers to the number of randomly and uniformly sampled articles from the inference dataset for each of the inference sample. Below, we summarize the F1 scores and the AUC scores that we have found from the different baselines and the proposed Invernet Framework for different N and B values.

Table 5.3: F1 and AUC scores for different N={5, 15, 30} and B={5, 25, 50} values for *AG_News* Dataset

Emb	Eval	n 5			n 15			**n 30**		
		b5	b25	b50	b5	b25	b50	b5	**b25**	b50
CBOW	F1	0.60	0.63	0.63	0.61	0.62	0.64	0.66	**0.69**	0.67
	AUC	0.71	0.73	0.74	0.73	0.76	0.77	0.75	**0.78**	0.76
Skip Gram	F1	0.59	0.60	0.62	0.62	0.65	0.66	0.67	**0.70**	0.68
	AUC	0.71	0.74	0.75	0.74	0.77	0.79	0.77	**0.79**	0.78
GloVe	F1	0.62	0.64	0.66	0.64	0.67	0.69	0.70	**0.72**	0.71
	AUC	0.74	0.76	0.76	0.75	0.77	0.79	0.77	**0.80**	0.79

In 5.3.4, we hypothesized that with a higher number of inference samples, our inference model will be exposed to a larger number of data with high variance and therefore will be able to learn better. Through our experiments, we validate our hypothesis. Table 5.3 is a reflection of the margin by which inference models trained with a higher number of samples perform better than inference models trained with fewer number of samples. In tables 5.3 and 5.4 we can observe that the best performance across different word embedding models was achieved with n=30 and b=25. However, we can notice an exception at n=30 and b=50 where testing F1 scores and AUC scores are better than most settings but slightly worse than n=30 and b=25. For each n^{th} sample, we are picking *b* random documents from the inference dataset and then replacing them into the inference pool. As a result, with a higher value of n and b, the likelihood of encountering duplicate documents across multiple inference samples increases. With such duplicity, the diversity in the inference sample distribution decreases thereby causing overfitting. On the other hand, with lower

values of n and b, we are sampling fewer documents and reducing the distributional diversity among the inference samples. So, the goal of tuning the hyperparameters n and b is to find optimal values of n and b so that we can get a diverse set of inference samples. In our experiments, empirically we have found n=30 and b=25 to enable the best performance of the inference model.

Table 5.4: F1 and AUC scores for different N={5, 15, 30} and B={5, 25, 50} values for *NYT* dataset

Emb	Eval	n 5			n 15			**n 30**		
		b5	b25	b50	b5	b25	b50	b5	**b25**	b50
CBOW	F1	0.59	0.62	0.64	0.65	0.66	0.68	0.67	**0.70**	0.68
	AUC	0.71	0.72	0.74	0.74	0.76	0.76	0.77	**0.79**	0.78
Skip Gram	F1	0.61	0.65	0.66	0.67	0.69	0.69	0.68	**0.71**	0.69
	AUC	0.71	0.73	0.74	0.74	0.77	0.78	0.78	**0.81**	0.79
GloVe	F1	0.64	0.66	0.68	0.67	0.69	0.70	0.69	**0.73**	0.71
	AUC	0.73	0.75	0.76	0.78	0.80	0.80	0.80	**0.82**	0.81

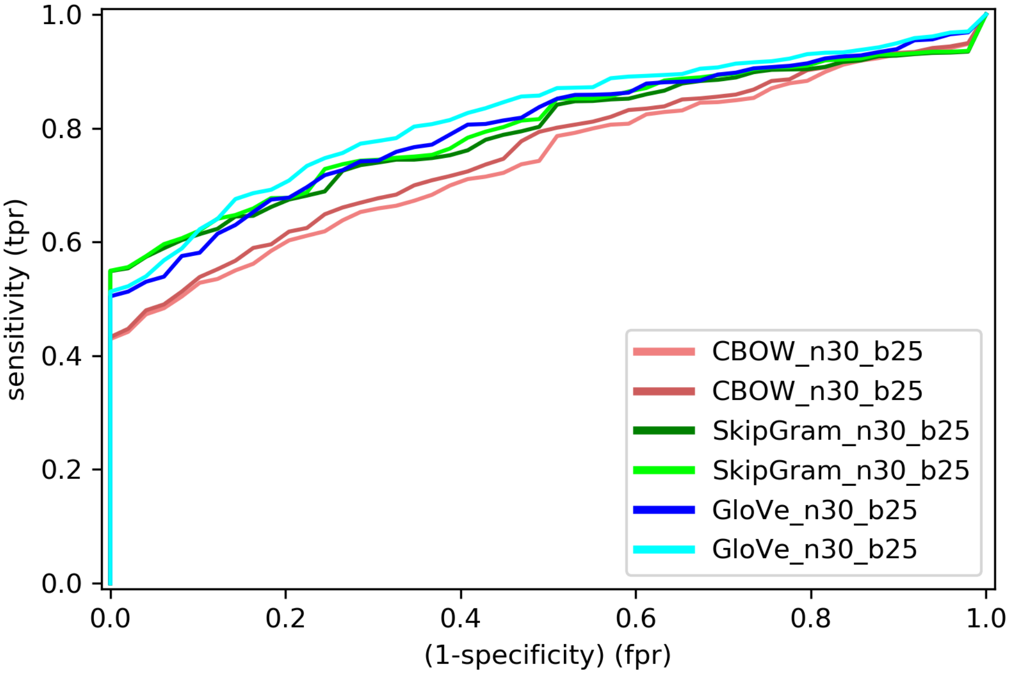


Figure 5.5: ROC analysis of Invernet Framework for different embedding models with **N=30** and **B=25** setting

5.5 Qualitative Analysis of Distributional Performance

We perform a heat map analysis to visualize the performance of our framework for target words with different frequency. In the figures below, we have the target vocabulary sorted by their frequency in both the x-axis and the y-axis. The color at cell (i, j) is closer to green if our inference model is able to correctly identify whether words in indices i and j co-occur together within each other's context. Otherwise, the color at cell (i, j) goes closer to red. The function governing the cell color of the following heat maps is as follows:

$$color_{(i,j)} = 1 - ||\hat{y}_{word(i),word(j)} - y_{word(i),word(j)}|| \tag{5.9}$$

In Eq. (5.9), if the value of *color* defines the distance between the ground truth $\hat{y}$ and the prediction y with respect to the value 1. The closer $\hat{y}$ and y is, the lower the distance and the higher the value of *color*. If the prediction is incorrect by a large margin, distance between $\hat{y}$ and y increases and the value of *color* decreases. With correct predictions, *color* is closer to 1 and the corresponding cell color becomes green. With incorrect predictions, *color* is closer to 0 and the corresponding cell color becomes red.

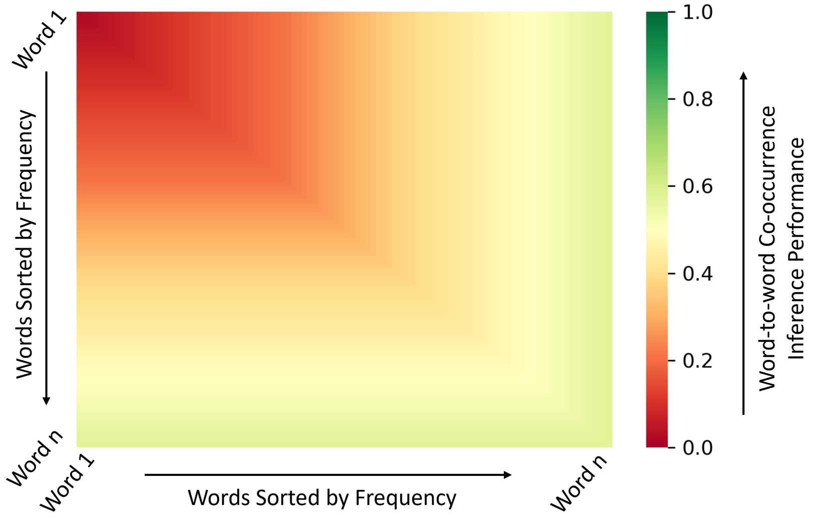


Figure 5.6: Word-to-word distributional performance of logistic regression based inference model on embeddings trained with CBOW model

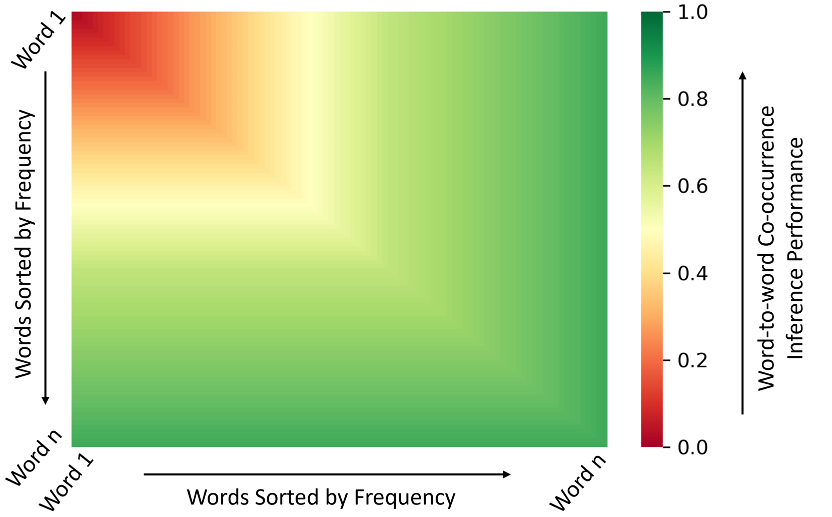


Figure 5.7: Word-to-word distributional performance of Invernet Framework on embeddings trained with CBOW model

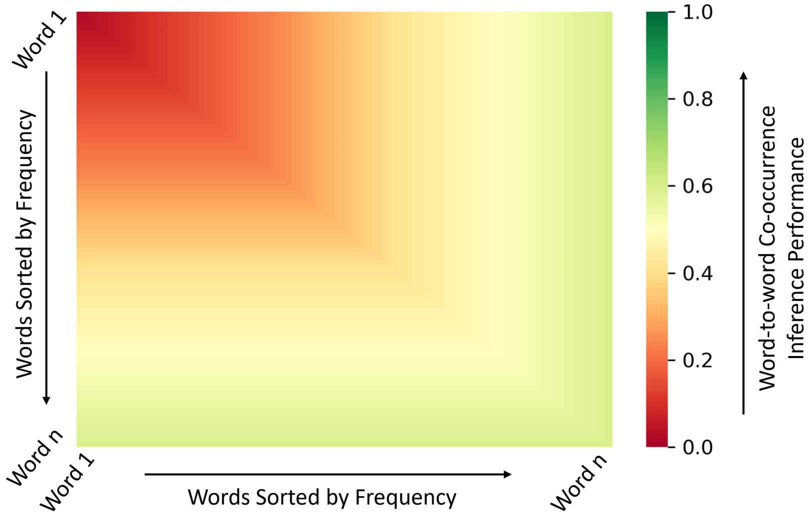


Figure 5.8: Word-to-word distributional performance of logistic regression based inference model on embeddings trained with SkipGram model

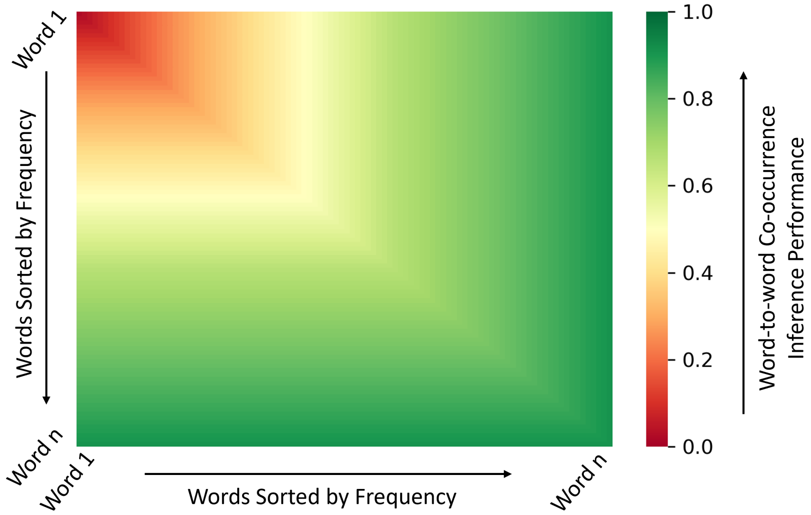


Figure 5.9: Word-to-word distributional performance of Invernet Framework on embeddings trained with SkipGram model

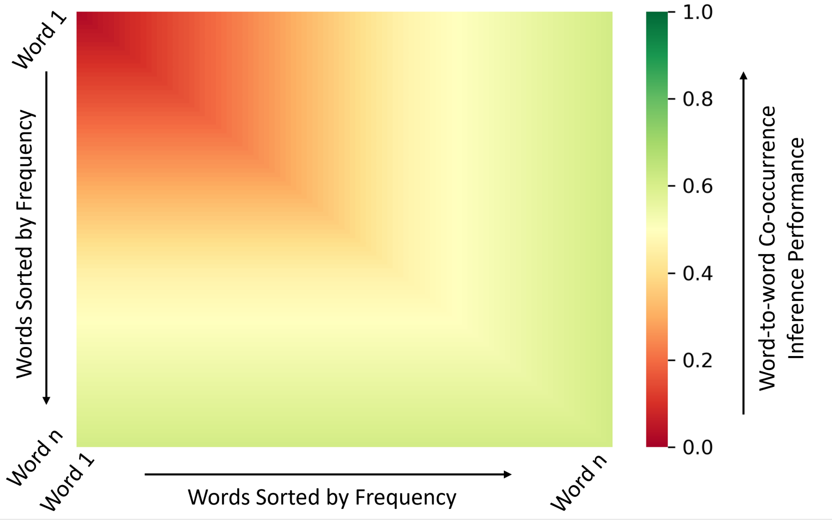


Figure 5.10: Word-to-word distributional performance of logistic regression based inference model on embeddings trained with GloVe model

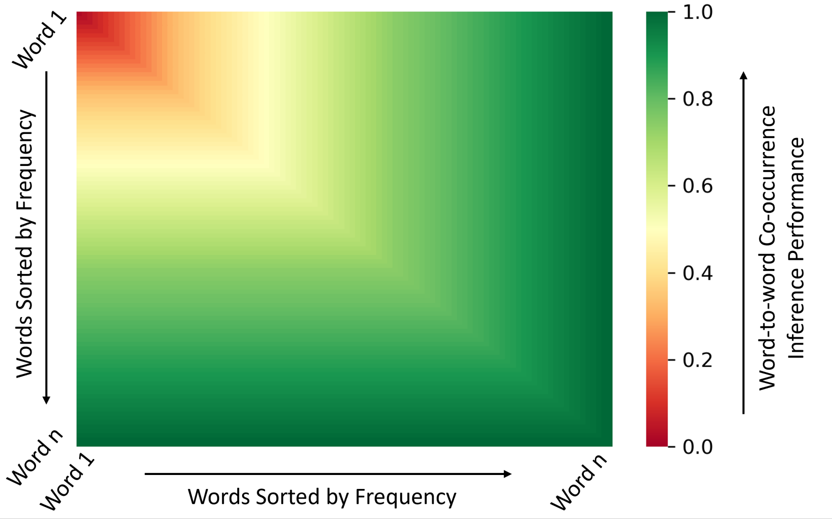


Figure 5.11: Word-to-word distributional performance of Invernet Framework on embeddings trained with GloVe model

Figures 5.6 and 5.7 show the distributional performance of the naive logistic regression baseline and the Invernet framework respectively for the CBOW (Continuous Bag-of-words) embedding model. Figures 5.8 and 5.9 show the distributional performance of the naive logistic regression baseline and the Invernet framework respectively for the SkipGram embedding model. Figures 5.10 and 5.11 show the distributional performance of the naive logistic regression baseline and the Invernet framework respectively for the GloVe embedding model.

For all three embedding models CBOW (Fig. 5.6), SkipGram (Fig. 5.8), and GloVe (Fig. 5.10), we can observe that the naive logistic regression baseline has a stark red region along the top left of the heat maps. At the same time, the naive logistic regression baseline produces a very light green region towards the bottom right of the heat maps. Such color pattern tells us that the naive logistic regression model is unable to identify the context of a large number of infrequent target vocabulary and that the logistic regression model shows very low confidence when identifying the context of frequent target vocabulary. On the other hand, for all three embedding models CBOW (Fig. 5.7), SkipGram (Fig. 5.9), and GloVe (Fig. 5.11), the Invernet framework shows light red regions towards the top left and a stark green region towards the bottom right. This color pattern tells us that, the Invernet framework is most of the times very confidently correct about the context of frequent target vocabulary and rarely incorrect with lower confidence regarding the infrequent target vocabulary. We can also objectively claim from the changes of pattern in Figs. 5.7, 5.9, and 5.11 that the performance of Invernet becomes progressively better from CBOW to SkipGram and from SkipGram to GloVe embedding model, confirming our hypothesis that the better a word embedding model becomes, the easier it becomes to invert those embeddings.

5.6 Membership Inference

In this experiment, we use the predicted co-occurrence vectors and try to infer whether any sentence or sentence fragment formed using such co-occurrence vectors is actually a part of the downstream dataset or not using a high level hit ratio analysis and a sequence reconstruction technique.

5.6.1 Quantitative Analysis with Hit Ratio

In the hit ratio analysis, we take some positive and negative documents from the target dataset (e.g. AG's News) and a non-target dataset (e.g. NYT Dataset) respectively. We form sets of 1 positive and 4 negative documents based on maximum vocabulary overlap. In this evaluation, our goal is to find out whether a given fine-tuned embedding is coming from the positive or negative documents based on the predicted binary context vector. The negative documents serve as the control of the experiment.

For each set of 1 positive and 4 negative documents, we consider the union of their vocabulary to form the context vectors. We compute the binary context vectors for each of the 5 documents from a set. These binary context vectors are formed with respect to a specific window size = 5 which means that only the immediately neighboring 5 left and 5 right words of the target word will have 1s in the binary context vector of that target word. Then, we apply the Invernet framework to the positive samples and predict the corresponding binary context vectors for the target words. Then we try to find whether we could hit or we missed the positive samples by comparing the corresponding binary context vectors. If the Hamming distance between the predicted context vectors and the positive sample's context vectors is less than that between the predicted and the negative samples' context vectors, we call it a hit and otherwise a miss. We consider top k retrieval for calling hits. For convenience we consider the relevance of a document as the inverse of the distance between the document's context vectors and the predicted vector. When k=2 for top k retrieval, if the relevance of the positive document is higher than (5-k)=3 negative documents, we call it a hit. When k=3 for top k retrieval, if the relevance of the positive document is higher than (5-k)=2 negative documents, we call it a hit.

We also perform a slightly relaxed hit ratio analysis by ranking the 5 documents from a specific set based on the inverse hamming distance between the predicted binary context vector of the positive document and the ground truth binary context vector of all the 5 documents. The lower the hamming distance the higher the document is ranked. In this case, the rank becomes the relevance score of the documents. We also have the ground truth ranking at disposal for reference.

Ideally, the positive document should always be ranked at the top. However, practically that is not always the case. Then we carry out a normalized discounted cumulative gain (NDCG) evaluation of the ranking.

Table 5.5: Comparing different samples sizes for hit ratio analysis

			Hit Ratio (Top k)			
Total	Positive	Negative	k=1	k=2	k=3	NDCG Score
250	50	200	0.51	0.53	0.55	0.58
500	100	400	0.53	0.54	0.57	0.60
1000	200	800	0.58	0.62	0.66	0.70
2000	400	1600	0.60	0.63	0. 69	0.73

Table 5.6: Hit Ratio Case Study

	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10
pred	1	1	0	0	1	0	0	1	1	0
pos	1	1	1	0	0	1	1	1	0	0
neg1	1	1	0	0	0	0	1	1	0	0
neg2	0	1	1	0	0	0	1	0	0	1
neg3	0	0	0	1	0	1	0	0	1	1
neg4	0	0	0	1	0	1	1	0	0	1

When ranking the positive and the negative samples with respect to how similar their binary context vectors are with that of the prediction vector, the positive sample should be at the top. However, from the results in Table 5.6, we have found the following ranking: neg2 ⟶ neg3 ⟶ pos ⟶ neg1 ⟶ neg4. We call this a hit since the positive sample is ranked higher than at least half of the negative samples. At the same time, we can see that the positive sample has dropped from first to third position in ranking because the binary context vectors of neg2 and neg3 samples had considerable similarity with the positive and the predicted binary context vectors. Nevertheless, because of using the ranking evaluation we can still consider this case as a hit.

5.6.2 Qualitative Analysis with Sequence Reconstruction

We apply the Invernet framework with an expanding window technique to infer an approximate reconstruction of the target sentence or a fragment of the sentence. When we create multiple inference samples for the Invernet framework, we usually create corresponding multiple context vectors. These ground truth context vectors are created with respect to a specific window size. If we incrementally iterate over a specific window length between the range of [1, ℓ] and at each step subtract the binary context vector from the previous window, we can sequentially reconstruct the target sentence fragments that are with reasonable confidence, members of the downstream dataset. An example sequence reconstruction from the AG's News dataset is as follows:

Table 5.7: Example of Sequence Reconstruction from AG's News Dataset

Citigroup acquires First American Bank

Table 5.8: Inference within window size W=1, target word='citigroup'

	citigroup	acquires	first	american	bank
$C_{W=1}$	0	1	0	0	0

Table 5.9: Inference within window size W=2, target word='citigroup'

	citigroup	acquires	first	american	bank
$C_{W=2}$	0	1	1	0	0
$C_{W=2}$ - $C_{W=1}$	0	0	1	0	0

Table 5.10: Inference within window size W=3, target word='citigroup'

	citigroup	acquires	first	american	bank
$C_{W=3}$	0	1	1	1	0
$C_{W=3}$ - $C_{W=2}$	0	0	0	1	0

Table 5.11: Inference within window size W=4, target word='citigroup'

	citigroup	acquires	first	american	bank
$C_{W=4}$	0	1	1	1	1
$C_{W=4}$ - $C_{W=3}$	0	0	0	0	1

Table 5.12: Final reconstructed sequence

target word	$C_{W=1}$	$C_{W=2}$ - $C_{W=1}$	$C_{W=3}$ - $C_{W=2}$	$C_{W=4}$ - $C_{W=3}$
citigroup	acquires	first	american	bank

Using the expanding window technique, we neither necessarily nor realistically need to reconstruct an exact sentence from the downstream dataset to breach user privacy. Instead, sensitive information surrounding certain events can be leaked from certain sequences of words that are predicted correctly. A simple justification about why this incrementally differential approach on binary context vector is useful to reconstruct specific word sequences or sentence fragments, can be obtained by comparing Tables 5.11 and 5.12. When we consider a large window size in Table 5.11 and infer the binary context vector of a target word only from a single large window size, we get which words appear in the context of a target word within the specified large window size but we are unable to infer the fine-grained positions of the words in the context. On the other hand, in Table 5.12, when we try to reconstruct the sequence using the expanding window technique and aggregate the incrementally differential binary context vectors, we get a sequence of words with granular leakage of the word positions surrounding the target word. Finally we can claim with reasonable certainty that the reconstructed sequence of words or sentence fragment is a member of the downstream target dataset. Evaluating the reconstructed sequence or fragment can be as simple as judging how much of the sequence matches with a one from the target dataset. Sometimes because of faulty predictions, there might be multiple contending words for the same position. Depending on the number of possibilities, we can manually pick the one that sounds more appropriate both grammatically and semantically. Another option to break the contention is to choose the word that occurs more frequently with the target word in the pretraining dataset since the pretraining dataset is our reference. We summarize our high level observations in the following table:

Table 5.13: Human-in-the-loop Similarity Judgement

Frequency of Observation	Similarity between Reconstructed Fragment and Actual Members
$\sim 55\%$ of the times	$\sim 50-55\%$
$\sim 25\%$ of the times	$> 60\%$
$\sim 20\%$ of the times	$<= 50\%$
	Expected Similarity $> 55\%$

Chapter 6

Conclusion

In this thesis, we have demonstrated that a significant amount of downstream context distribution can be leaked through fine-tuned word embeddings if one has access to the pretrained word embeddings, pretraining dataset, and the fine-tuned word embeddings. We have empirically shown that a successful embedding inversion attack can be carried using the Invernet framework with an attack F1 score of 0.70 and an attack AUC score of 0.79. Such performance metrics prove that Invernet can considerably exceed comparable performance of other methods that are mentioned in the literature. Through various comparative ablation studies We have established that word embedding models that are considered superior in various Natural Language Processing tasks, tend to be more vulnerable to embedding inversion attacks. A distributional heat map analysis has shown that embeddings of words that occur more frequently in the dataset generally leak the highest amount of context distribution.

We have noted from the literature and from common practices in the industry that small and large scale users alike resort to transfer learning for fine-tuning a set of pretrained word embeddings in order to reduce the computational overhead and carbon footprint of training from scratch. We have also observed a proliferation of publicly shared fine-tuned word embeddings. A primary reason for sharing or storing such fine-tuned word embeddings and word embedding models is to contribute very topical word embeddings to niche communities like the finance sector, the biomedical domain etc. for improving domain specific tasks in these sectors. However, a critical concern surrounds how these fine-tuned word embeddings are shared. Because the word embeddings are simply dense continuous vector representations of words, the publishers of such fine-tuned word embeddings are most of the times not careful about the potential of information leakage from these presumably unexplainable and uninvertible sets of vectors. So, numerous such fine-tuned word embeddings are shared or stored in an insecure way.

Our study focuses on the specific circumstances when a malicious participant in a transfer learning based Natural Language Processing system has access to both the pretraining dataset and pretrained word embeddings either through ownership rights or from the public domain. We assume that the attacker also gets access to a set of downstream embeddings that are fine-tuned on another user's private dataset. The attacker can access the fine-tuned word embeddings from the public domain if the owner of the fine-tuned word embeddings shares those embeddings in public. The attacker might also be able to access the fine-tuned word embeddings from a private domain if proper security and encryption mechanisms are not in place. We have also demonstrated with considerable confidence that under the aforementioned situation, an attacker can easily apply the Invernet framework to infer the context distribution of the downstream dataset. In light of our findings regarding information leakage from fine-tuned word embeddings, we wish to raise awareness about taking care when storing or sharing such fine-tuned word embedding models.

6.1 Future Directions

We have identified a few promising future directions of research arising from our work. In this section, we will briefly mention these potential future directions of research in the field of fine-tuned embedding inversion.

In this work, we have presented how the co-occurrence distribution of a fine-tuning dataset can be inferred with access to both the pretrained and fine-tuned embeddings and with the knowledge of the pretraining dataset. We have reported the successful outcomes of the Invernet framework in various unsupervised fine-tuning settings. Another interesting direction can be to explore the potential context distribution leakage of downstream embeddings that are fine-tuned in a supervised setting with respect to a specific downstream task like text classification, next word/sentence predictions, named entity recognition etc.

A critical step in the Invernet framework is to create multiple inference samples of various documents from a separate inference dataset taken from the original pretraining dataset. We generate multiple sets of word embeddings and corresponding binary co-occurrence vectors for each target word from the multiple inference samples. The inference model eventually learns the relation between these word embeddings and binary co-occurrence vectors. One can augment the inference samples by using a text generation module. The text generation module will essentially generate numerous text samples based on the inference dataset. As a result, with a huge number of inference samples, the augmented Invernet framework should hypothetically perform even better.

The output of the Invernet framework gives us word-to-word co-occurrence context of the downstream dataset. If one passes each word from the vocabulary as the target word through the Invernet framework, it is possible to get a probability score of two words occurring together within a certain window size. Eventually, one can use the realized word-to-word co-occurrence probability scores and apply any probabilistic text generation method like a Hidden Markov Model (HMM) incorporated with knowledge from a Part-of-Speech (POS) tagger or linguistic grammar to generate text equivalent to the actual downstream dataset. Successful dataset inversion through this technique can pose a serious threat to user privacy.

References

[1] Glove: Global vectors for word representation.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org/`, 2015. Software available from tensorflow.org.

[3] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063*, 2019.

[4] Karlo Babić, Sanda Martinčić-Ipšić, and Ana Meštrović. Survey of neural text representation models. *Information*, 11(11):511, 2020.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[6] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in Neural Information Processing Systems*, 13, 2000.

[7] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Cen Chen, Bingzhe Wu, Minghui Qiu, Li Wang, and Jun Zhou. A comprehensive analysis of information leakage in deep transfer learning. *arXiv preprint arXiv:2009.01989*, 2020.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] Nicholas Dingwall and Christopher Potts. Mittens: an extension of glove for learning domain-specialized representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 212–217, 2018.

[12] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016.

[15] Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1):1–23, 2021.

[16] Antonio Gulli. Ag's corpus of news articles. `http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html`.

[17] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[19] Tom Kocmi and Ondřej Bojar. An exploration of word embedding initialization in deep-learning tasks. *arXiv preprint arXiv:1711.09160*, 2017.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[21] Tomas Mikolov and et al. word2vec, 2013.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[23] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[24] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[25] Kevin Patel and Pushpak Bhattacharyya. Towards lower bounds on number of dimensions for word embeddings. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 31–36, 2017.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher,

M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[28] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas. A taxonomy and survey of attacks against machine learning. *Computer Science Review*, 34:100199, 2019.

[29] Radu Raicea. Want to know how deep learning works? here's a quick guide for everyone., Oct 2017.

[30] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[31] Sebastian Ruder. *Neural transfer learning for natural language processing*. PhD thesis, NUI Galway, 2019.

[32] Sebastian Ruder. Recent Advances in Language Model Fine-tuning. `http://ruder.io/recent-advances-lm-fine-tuning`, 2021.

[33] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*, pages 15–18, 2019.

[34] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[35] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[36] Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 377–390, 2020.

[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[38] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] Yuxuan Wang, Yutai Hou, Wanxiang Che, and Ting Liu. From static to dynamic word representations: a survey. *International Journal of Machine Learning and Cybernetics*, 11(7):1611–1630, 2020.

[41] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[42] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[43] Donna Xu, Yaxin Shi, Ivor W Tsang, Yew-Soon Ong, Chen Gong, and Xiaobo Shen. Survey on multi-output learning. *IEEE transactions on neural networks and learning systems*, 31(7):2409–2429, 2019.

[44] Zijun Yao, Yifan Sun, Weicong Ding, Nikhil Rao, and Hui Xiong. Dynamic word embed-

dings for evolving semantic discovery. In *WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining*, 2018.

[45] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NIPS*, 2015.