



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Bachelor's Thesis

Influence of Classification Model Architecture on Anomaly Detection in Text

Tommaso Gargiani
Open Informatics

May 2021

<https://github.com/tgargiani/OOD-anomaly-detection-in-text>

Supervisor: Ing. Petr Lorenc

I. Personal and study details

Student's name: **Gargiani Tommaso** Personal ID number: **483466**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Open Informatics**
Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Influence of Classification Model Architecture on Anomaly Detection in Text

Bachelor's thesis title in Czech:

Vliv architektury klasifikačního modelu na detekci anomálií v textu

Guidelines:

The bachelor thesis will comprise the following steps:

1. Review the possible algorithms for anomaly detection based on different neural network architectures, focusing on:
 - last layer [1,5]
 - loss function [2,3,4]
2. Review the possible datasets for anomaly detection
 - Must include CLINC150, ROSTD and dataset provided by supervisor
3. Implement some of the state-of-the-art algorithms
4. Compare selected approaches in respect to false rejection rate, precision, memory and time requirements on chosen datasets

Bibliography / sources:

- [1] Lei Shu, Hu Xu, Bing Liu – DOC: Deep Open Classification of Text Documents – University of Illinois at Chicago – 2017.
- [2] Ting-En Lin, Hua Xu – Deep Unknown Intent Detection with Margin Loss – Tsinghua University, Beijing, China – 2019.
- [3] Feng Wang, Weiyang Liu, Haijun Liu, Jian Cheng – Additive Margin Softmax for Face Verification – UESTC, Chengdu, China – 2018.
- [4] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, Wei Liu – CosFace: Large Margin Cosine Loss for Deep Face Recognition – Tencent AI Lab, Bellevue, USA – 2018.
- [5] Hanlei Zhang, Hua Xu, Ting-En Lin – Deep Open Intent Classification with Adaptive Decision Boundary – Tsinghua University, Beijing, China – 2020.

Name and workplace of bachelor's thesis supervisor:

Ing. Petr Lorenc, Department of Cybernetics, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **08.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Petr Lorenc
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgement / Declaration

First of all, I would like to thank my supervisor Ing. Petr Lorenc for his careful guidance and valuable advice. I also thank my family for always supporting me throughout my education and for being close to me, no matter the distance between us.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 21, 2021

.....

Abstrakt / Abstract

Konverzační agenti pracují v různých prostředích. V open world prostředí hraje důležitou roli jak rozpoznávání předdefinovaných in-domain intentů, tak detekce neznámých out-of-domain anomálií. V této práci zkoumáme různé metody určené pro detekci anomálií a představujeme novou metodu, která je nezávislá na out-of-domain datech. Metody jsou následně porovnány s ohledem na in-domain přesnost, out-of-domain recall, false rejection rate a časové a paměťové požadavky. Ohodnocení metod na několika datasetech ukazuje, že námi představená metoda výrazně překonává již zavedené metody.

Klíčová slova: klasifikace intentů, detekce out-of-domain, detekce anomálií, chatbot, zpracování přirozeného jazyka

Překlad titulu: Vliv architektury klasifikačního modelu na detekci anomálií v textu

Conversational agents operate in various environments. In the open world environment, both recognition of predefined in-domain intents and detection of unknown out-of-domain anomalies play a crucial role. In this thesis, we review several methods for anomaly detection and propose a novel method that is independent of out-of-domain data. The methods are then compared in respect to in-domain accuracy, out-of-domain recall, false rejection rate and time and memory requirements. Evaluations on a variety of datasets show that Our Proposed Method significantly outperforms the current state-of-the-art.

Keywords: intent recognition, out-of-domain detection, anomaly detection, chatbot, natural language processing

/ Contents

1 Introduction	1
2 Datasets	2
2.1 CLINC150	2
2.2 ROSTD	2
2.3 Supervisor Dataset	3
3 Neural Networks	4
4 Loss Functions	6
4.1 Softmax Loss	6
4.2 Large Margin Cosine Loss	6
4.3 Triplet Loss	7
5 Embeddings	9
5.1 Universal Sentence Encoder	9
5.1.1 USE-DAN	9
5.1.2 USE-TRAN	9
5.2 Sentence-BERT	9
5.3 Metric Learning	10
6 Methods	11
6.1 Cosine Similarity	11
6.2 Baseline Neural Network	11
6.3 DeepUnk	11
6.3.1 Local Outlier Factor	12
6.4 Adaptive Decision Boundary	12
6.5 Proposed Method	13
7 Experiments	15
7.1 Metrics	15
7.1.1 Precision and Recall	15
7.1.2 FRR	15
7.1.3 Formulae	15
7.2 Experimental Setup	16
7.3 ADB Performance	16
7.4 CLINC150	17
7.5 ROSTD	18
7.6 Supervisor Dataset	19
7.7 Limited Sentences and Intents	19
8 Conclusion	22
References	23

Tables / Figures

2.1	CLINC150 dataset splits.....	2	1.1	Example of user-chatbot interaction	1
2.2	ROSTD dataset splits.....	2	3.1	Connection between biological neurons	4
2.3	Supervisor Dataset overview.....	3	3.2	Neural network	4
7.1	Original ADB performance	16	4.1	Comparison of LMCL with Softmax Loss	7
7.2	New ADB performance	16	4.2	Triplet Loss learning	8
7.3	Results on CLINC150.....	17	5.1	Sentence-BERT siamese model structure	10
7.4	Time and memory on CLINC150	17	6.1	Stopping criterion of Our Proposed Method	14
7.5	Metric learning time on CLINC150	17	6.2	Effect of metric learning and Our Proposed Method on toy data	14
7.6	Results on ROSTD.....	18	7.1	Comparison between Our Proposed Method and ADB on 10 intents	20
7.7	Time and memory on ROSTD .	18	7.2	Comparison between Our Proposed Method and ADB on 25 intents	20
7.8	Metric learning time on ROSTD	18	7.3	Comparison between Our Proposed Method and ADB on 50 intents	21
7.9	Results on Supervisor Dataset .	19			
7.10	Time and memory on Supervisor Dataset	19			
7.11	Metric learning time on Supervisor Dataset.....	19			

Chapter 1

Introduction

Intent classification is crucial to conversational AI, a branch of Natural Language Processing (NLP), as it allows the chatbot to recognize what the user intends to do. Chatbots, i.e. systems like Amazon’s Alexa or Apple’s Siri, heavily rely on conversational-centric (CC) style [1]. Even though the CC style is expensive to design, its better dialogue management and improved coherence outweigh the disadvantages [2–3].

The system depends on recognition of predefined *in-domain* (ID) intents. Examples of sentences that belong to an ID class are provided by a dialogue designer. These examples are then used in a classification model that is trained to classify sentence user input into one of the intents. Therefore, thanks to intent recognition, the chatbot is able to guide the user through a predefined conversation flow.

However, based on the open world assumption [4], we need to prepare the system for situations when the user input does not belong to any known in-domain class. These inputs are called *out-of-domain* (OOD, *anomalies*). Unfortunately, while classifiers generally perform very well on in-domain classification, they tend to struggle on OOD detection.

A chatbot that fails to recognize an out-of-domain query will give an unrelated answer, instead of using a fallback response. An example of this can be seen in Figure 1.1.



Figure 1.1. Example of interaction between user (right side) and chatbot (left side). (1) – the user asks a question that belongs to an ID intent. (2) – the chatbot fails to detect OOD. (3) – the chatbot succeeds in the task, giving a fallback response [5].

In this thesis, we will focus on the methods employed to classify correctly both in-domain and out-of-domain intents, aiming to improve the usability of conversational agents.

Chapter 2

Datasets

A chatbot has to handle many different forms of user input, as not everyone speaks in the same manner. For instance, User A may talk to the chatbot in long, formal sentences, while User B might speak more concisely. Different chatbot purposes should also be taken into consideration – a conversational chatbot like Alquist [3] will need different sentence examples than a Q&A chatbot.

Therefore, to reflect all the varieties of user input, we evaluate the employed models on 3 different datasets – CLINC150, ROSTD and a dataset provided by our supervisor.

2.1 CLINC150

CLINC150 [5] contains 23,700 labeled sentences, out of which 22,500 are in-domain, covering 150 intents, and 1,200 are out-of-domain. The exact number of sentences per in-domain intent of either the training, validation or test split can be seen in Table 2.1. The out-of-domain splits are also shown.

Training		Validation		Test	
ID	OOD	ID	OOD	ID	OOD
100	100	20	100	30	1,000

Table 2.1. Training/validation/test splits of each in-domain intent and of out-of-domain.

This dataset contains rather long sentences, spanning a range of intents that go from *translation* to *groceries*.

2.2 ROSTD

ROSTD [6] is a companion to the in-domain dataset by [7], ROSTD extends the aforementioned ID dataset with more than 4,000 OOD sentences.

The in-domain dataset features 12 intents with more than 43,000 examples. However, many of these intents resemble each other, as they cover only 3 domains – *Alarm*, *Reminder* and *Weather*. The number of sentences of the training, validation and test splits of each domain, as well as out-of-domain, is described in Table 2.2.

Domain	Training	Validation	Test	Number of intents
Alarm	9,282	1,309	2,621	6
Reminder	6,900	943	1,960	3
Weather	14,339	1,929	4,040	3
OOD	750	750	3,090	1

Table 2.2. Training/validation/test splits and number of intents of every domain.

ROSTD’s example sentences fit a Q&A chatbot that is able to execute IoT commands.

2.3 Supervisor Dataset

The unnamed dataset provided by our supervisor is as yet unpublished. Due to its chatbot-focused design, it is not comparable with the two previously described datasets.

It features 81 dialogues on 13 different topics. Each dialogue has several decision points which limit the current selection of *local* intents. A dialogue also contains *global* intents, i.e. intents accessible from every decision point. Out-of-domain examples are the same for all dialogues.

An overview of the dataset is available in Table 2.3.

Item	Number
Topics	13
Dialogues	81
Avg. decision points per dialogue	3.5
Avg. local intents per decision point	2.8
Avg. global intents per dialogue	17.9

Table 2.3. Supervisor Dataset overview.

Chapter 3

Neural Networks

A neural network (NN) is a model used to classify and cluster data. It is composed of units that are inspired by the biological neurons in our brain. These units are organized in fully connected layers – every unit in a layer is connected to all the units in both the previous and next layers. Such connections mimic the function of synapses between biological neurons (Figure 3.1), allowing to transmit the output of a unit, a real number, to the next layer's units. This type of NN with no backward connection is called a *feedforward neural network*.

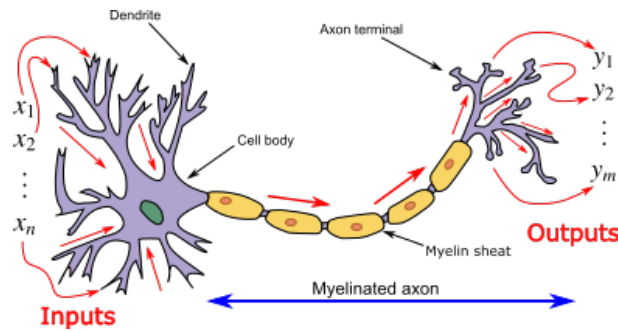


Figure 3.1. Connection between biological neurons [8].

According to the Universal Approximation Theorem [9], neural networks can be used to approximate any continuous function on a unit hypercube.

Every neural network has an input and output layer, with zero or more hidden layers in between, as visualized in Figure 3.2.

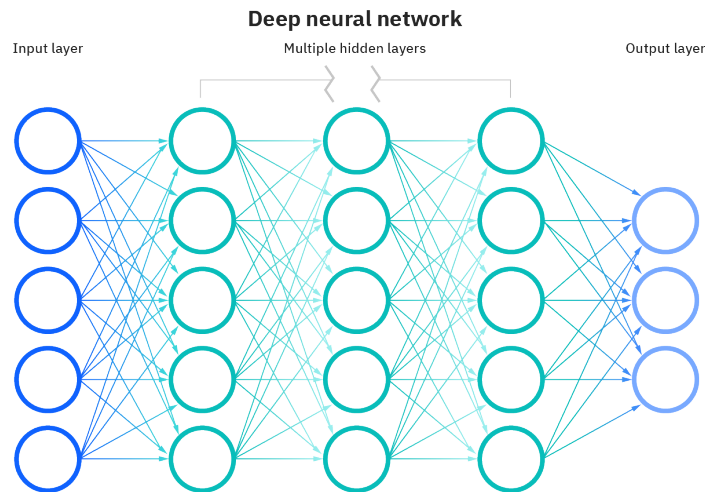


Figure 3.2. A neural network with 3 hidden layers [10].

The units in the hidden and output layers have associated weights, a bias and an activation function. Activation functions are also called *non-linearities* due to their non-linear properties that allow NNs to cope with more complex problems, where a linear function would not suffice. Common activation functions include:

- Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Tanh

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Softmax

$$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}} \text{ for } i = 1, 2, \dots, K$$

- ReLu

$$f(x) = \max(0, x)$$

The output $x \in \mathbb{R}$ of a unit is defined as:

$$x = \sigma(\mathbf{w}^T \mathbf{z} + b)$$

where $\mathbf{w} \in \mathbb{R}^N$ are the weights associated with the unit, $\mathbf{z} \in \mathbb{R}^N$ are the outputs of the previous layer, $b \in \mathbb{R}$ is the bias and σ is the activation function.

The output $\mathbf{x} \in \mathbb{R}^M$ of a layer with M units can be thus generalized to:

$$\mathbf{x} = \sigma(\mathbf{W}\mathbf{z} + \mathbf{b})$$

where $\mathbf{W} \in \mathbb{R}^{M \times N}$ is the weights matrix, $\mathbf{z} \in \mathbb{R}^N$ are the outputs of the previous layer, $\mathbf{b} \in \mathbb{R}^M$ is the bias vector and σ is the activation function.

We can observe that a neural network with L layers (excluding the input layer) is a composition of multiple functions:

$$\mathbf{x} = \sigma_L(\mathbf{W}_L(\cdots(\sigma_1(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1))) + \mathbf{b}_L)$$

where $\mathbf{x} \in \mathbb{R}^M$ is the output and $\mathbf{z} \in \mathbb{R}^N$ is the input of the neural network.

The weights and bias of a unit are parameters that are learned by the network during the training phase. In fact, training can be formulated as an optimization problem where a loss function is minimized by updating the NN's weights and biases.

Loss functions are often minimized using gradient-based methods. Gradient descent finds a local minimum of the differentiable function f by taking steps, of size learning rate μ , in the opposite direction of the gradient:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mu \nabla f(\mathbf{x}_k)$$

Chapter 4

Loss Functions

In this chapter, we will discuss the loss functions that were minimized by the neural networks we employed.

4.1 Softmax Loss

The term *Softmax Loss* is often used for the combination of the Softmax activation function and the Cross-Entropy Loss.

Softmax is a function that takes the output $\mathbf{x} \in \mathbb{R}^K$ of the last layer of a classification neural network with K classes, i.e. numbers called *logits*, and transforms them into a probability distribution that sums to 1. This is achieved by applying the exponential function to the output of a class $x_i \in \mathbb{R}$ and then dividing the result by the sum of all the exponentials:

$$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}}$$

Then, this probability distribution is used by Cross-Entropy Loss to measure the performance of the classification model:

$$CEL = \frac{1}{N} \sum_{i=1}^N -\log f(\mathbf{x})_{t_i}$$

where N is the number of training examples and t_i is the ground truth class of an example.

4.2 Large Margin Cosine Loss

Large Margin Cosine Loss was first used as part of the CosFace model in [11].

LMCL builds upon the Softmax Loss and aims to minimize intraclass variance and maximize interclass variance by normalizing both features and weights vectors.

Let us rethink the class output x_i of a model's last layer in a cosine perspective:

$$x_i = \mathbf{w}_i^T \mathbf{z} = \|\mathbf{w}_i\| \|\mathbf{z}\| \cos \theta_i$$

where \mathbf{w}_i is the weights vector of a certain class, \mathbf{z} is the layer's input vector and θ_i is the angle between these two vectors.

We can observe that both the norm and the angle between vectors contribute to the class output x_i .

In order to avoid disruptive effects on feature learning, we normalize the weights vector \mathbf{w}_i . Then, variations in radial direction are removed by fixing $\|\mathbf{z}\|$ to the hyperparameter s . Thanks to this limitation, the learned features are separable in the

angular space. Finally, the hyperparameter $m \geq 0$ is used to reinforce a more stringent cosine margin.

The resulting loss is defined as:

$$LMCL = \frac{1}{N} \sum_{i=1}^N -\log \frac{e^{s(x_{t_i}-m)}}{e^{s(x_{t_i}-m)} + \sum_{j \neq t_i}^K e^{s x_j}}$$

subject to

$$\begin{aligned} \mathbf{w} &= \frac{\mathbf{w}}{\|\mathbf{w}\|} \\ \mathbf{z} &= \frac{\mathbf{z}}{\|\mathbf{z}\|} \\ x_{t_i} &= \mathbf{w}_{t_i}^T \mathbf{z} \end{aligned}$$

where N is the number of examples, K is the number of classes, s and m are hyperparameters, t_i is the ground truth class of an example, \mathbf{w}_{t_i} is the weights vector of the class, \mathbf{z} is the input vector and x_{t_i} is the output of the class.

A comparison of LMCL with Softmax Loss using multiple values of m is shown in Figure 4.1.

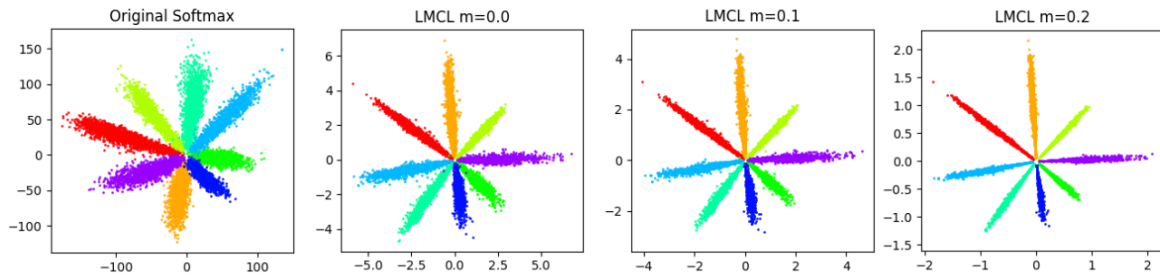


Figure 4.1. Comparison of LMCL with Softmax Loss in Euclidean space [11].

4.3 Triplet Loss

Triplet Loss [12] works with the normalized feature vector $\mathbf{x} \in \mathbb{R}^D$, where D is the vector dimension.

It ensures that a vector \mathbf{x}_a , the *anchor*, is closer to another vector of the same class \mathbf{x}_p , the *positive*, than to the vector of a different class \mathbf{x}_n , the *negative*. These three vectors combined form a triplet.

The loss for a single triplet is defined as:

$$TL = \max(\|\mathbf{x}_a - \mathbf{x}_p\| - \|\mathbf{x}_a - \mathbf{x}_n\| + \alpha, 0)$$

subject to

$$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

where α is a hyperparameter that defines the margin between positive and negative pairs.

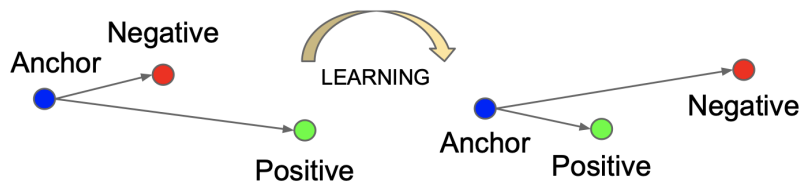


Figure 4.2. Triplet Loss learning [12].

Triplet Loss learning is visualized in Figure 4.2.

Proper triplet selection is critical for both effective learning and fast convergence. While there are several strategies, we use the *semi-hard triplet* strategy, as it gave the best results in the original paper. Semi-hard triplets are vectors where the resulting loss is still positive, albeit the negative is not closer to the anchor than the positive:

$$\|\mathbf{x}_a - \mathbf{x}_p\| < \|\mathbf{x}_a - \mathbf{x}_n\| < \|\mathbf{x}_a - \mathbf{x}_p\| + \alpha$$

Chapter 5

Embeddings

In NLP, sentence embeddings are used to represent sentence features in a high-dimensional vector format. Moreover, the embedding vector space is designed such that semantically similar words have embeddings that are closer to each other.

We have used 3 different types of embeddings – Universal Sentence Encoder with Deep Average Network (USE-DAN), Universal Sentence Encoder with Transformer Architecture (USE-TRAN) and Sentence-BERT (SBERT).

5.1 Universal Sentence Encoder

According to [13], Universal Sentence Encoder embeddings are trained on both supervised and unsupervised data. Supervised learning is performed with data from the Stanford Natural Language Inference (SNLI) corpus [14], whereas unsupervised data is collected from several web sources such as Wikipedia, discussion forums, etc.

5.1.1 USE-DAN

This encoding model creates sentence embeddings by means of a deep averaging network (DAN) that averages word and bi-gram embeddings and passes them through a feedforward deep neural network.

USE-DAN's time complexity is linear in the sentence length, hence its inference is efficient. Conversely, a drawback of having an efficient model is its slightly reduced accuracy.

5.1.2 USE-TRAN

As stated by [13], USE-TRAN creates sentence embeddings using the encoding sub-graph of the transformer architecture [15]. Relying on this solution enables the encoding model to compute context aware word embeddings that are later converted to sentence embeddings. Such conversion is applied by calculating the element-wise sum of the word embeddings at each word position.

In comparison with USE-DAN, USE-TRAN's complex nature allows the model to achieve higher accuracy at the cost of a penalized compute time and memory usage. Both these variables scale dramatically with sentence length.

5.2 Sentence-BERT

Sentence-BERT [16] is a modification to BERT [17], a pre-trained transformer network. Although BERT's contextualized embeddings set new state-of-the-art results to several NLP tasks, its large computational requirements make BERT unsuitable for tasks like semantic similarity comparison and clustering. Besides this, BERT does not directly compute stand-alone sentence embeddings, although this limitation can be avoided by averaging its outputs, for instance.

Firstly, BERT is fine-tuned on siamese and triplet networks [12] in order to ensure that the resulting embeddings are semantically meaningful and comparable with cosine similarity. Then, Sentence-BERT adds a pooling operation to the output of BERT in order to extract a sentence embedding.

The exact training dataset and model structure depend on the task the model was trained for. We use the *stsb-roberta-base* model which was optimized for Semantic Textual Similarity. This model computes the mean of all output vectors as its pooling strategy and then applies the softmax function to the difference between two sentence embeddings. Finally, cross-entropy loss is optimized. This structure is visualized in Figure 5.1.

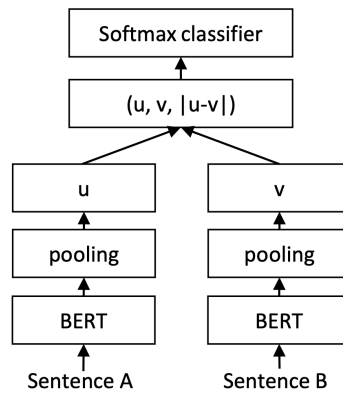


Figure 5.1. Sentence-BERT siamese model structure [16].

5.3 Metric Learning

As shown in [18], the main goal of metric learning is to learn a function f for which vectors $x_1, x_2, \dots, x_n \in Y_A$ and vectors $z_1, z_2, \dots, z_m \in Y_B$ are transformed such that $d(x_1, x_2) < d(x_1, z_1)$, where $d(x, y)$ is a distance function.

In other words, metric learning is used to minimize intra-class variance and maximize inter-class variance of the embeddings by training a neural network that approximates the function f .

Such modified embeddings should better reflect the prior knowledge we have about the known in-domain intents.

Chapter 6

Methods

In this chapter, we will introduce the several methods that we employed – Cosine Similarity, Baseline Neural Network (BaselineNN), DeepUnk, Adaptive Decision Boundary (ADB) and a novel method that we propose.

Two of these methods work with out-of-domain training data – Cosine Similarity and Baseline Neural Network. However, OOD data collection is a very challenging task, as it is impossible to reflect all the possible out-of-domain sentences that the user might say. Also, one change in the defined in-domain intents, e.g. adding a new intent, may entail many changes in the OOD training set. Therefore, we also experiment with methods that do not need OOD training data – DeepUnk, ADB and the novel proposed method.

6.1 Cosine Similarity

Cosine Similarity is a simple and straightforward method to measure semantic similarity between two embeddings.

Given two embeddings $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$, where D is the embedding dimension, Cosine Similarity can be computed as:

$$similarity = \cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

The formula above yields similarities ranging from -1 to 1 , where 1 means complete similarity and -1 complete dissimilitude.

6.2 Baseline Neural Network

BaselineNN is a simple neural network with no hidden layers. Softmax Loss is minimized. As mentioned before, out-of-domain examples are treated as an additional in-domain intent during training.

6.3 DeepUnk

This method was presented by [19] and adapted to our needs.

First, CosFace, a model with one hidden layer and with the Large Margin Cosine Loss, is built. Thanks to the use of LMCL, CosFace has not only learned to classify the in-domain classes, but has also strengthened, in terms of minimizing intraclass and maximizing interclass variance, the discriminative features of the embeddings.

The improved embeddings are thus extracted from the hidden layer and fed into Local Outlier Factor (LOF) [20], an algorithm that is able to detect OOD.

Finally, CosFace and LOF are combined in the prediction process. CosFace first classifies test examples into in-domain intents. Then, the embeddings learned with LMCL are extracted and LOF detects the out-of-domain examples.

6.3.1 Local Outlier Factor

As mentioned before, LOF is an algorithm capable of detecting anomalies (outliers), in our case out-of-domain sentences. This is achieved by identifying examples that have a significantly lower local density than their neighbors.

To measure the local outlier factor of an object A , i.e. the degree to which A is an outlier, we have to first introduce the reachability distance of the object A with regard to object B :

$$\text{reach-dist}_k(A, B) = \max(k\text{-dist}(B), d(A, B))$$

where $k\text{-dist}(B)$ is the distance between object B and its k -th nearest neighbor, and $d(A, B)$ is the distance between objects A and B . Reachability distance is used to introduce a smoothing effect in case objects A and B are considered too close to each other.

Then, we calculate local reachability density of object A , $\text{lrd}_k(A)$, as the inverse of the average reachability distance:

$$\text{lrd}_k(A) = \frac{|N_k(A)|}{\sum_{B \in N_k(A)} \text{reach-dist}_k(A, B)}$$

where $N_k(A)$ is the set of k -nearest neighbors. A low local reachability density is a sign that the object A is in a sparse area, hence more distant from other objects.

Local Outlier Factor is computed as:

$$\text{LOF}_k(A) = \frac{\sum_{B \in N_k(A)} \frac{\text{lrd}(B)}{\text{lrd}(A)}}{|N_k(A)|}$$

A value of LOF significantly larger than 1 indicates that the measured object is an outlier.

6.4 Adaptive Decision Boundary

ADB [21] is a two-step algorithm based on metric learning and creation of decision boundaries specific to each class using a neural network.

First, Softmax Loss metric learning is used to adapt embeddings to the in-domain intents.

Then, we create spherical boundaries around the centroid of each class. A spherical boundary aims to contain all examples of an in-domain class. On the other hand, if an example does not belong to the decision boundary of the closest centroid, it is considered as out-of-domain.

The centroid $\mathbf{c}_k \in \mathbb{R}^D$, where D is the embedding dimension, of class k is defined as:

$$\mathbf{c}_k = \frac{\sum_{\mathbf{x} \in \mathbf{X}_k} \mathbf{x}}{|\mathbf{X}_k|}$$

where \mathbf{X}_k contains all the training embeddings of class k .

Formally, an example \mathbf{x} belongs to class k if the following constraint is satisfied:

$$\|\mathbf{x} - \mathbf{c}_k\| \leq r_k$$

where $\|\mathbf{x} - \mathbf{c}_k\|$ denotes the Euclidean distance between \mathbf{x} and \mathbf{c}_k , and r_k is the class's radius.

Based on the suggestion of [22], the Softplus activation function is used to map between r_k and the actual radius learned by the model \hat{r}_k :

$$r_k = \log(1 + e^{\hat{r}_k})$$

Although a spherical boundary aims to contain all the in-domain examples of a certain class, this is not often feasible. Therefore, the proposed *Boundary Loss* is minimized. Boundary Loss balances the radius of each boundary so that it contains as many in-domain examples as possible, without sacrificing out-of-domain detection performance:

$$BL = \frac{1}{N} \sum_{i=1}^N [\delta_i (\|\mathbf{x}_i - \mathbf{c}_{t_i}\| - r_{t_i}) + (1 - \delta_i)(r_{t_i} - \|\mathbf{x}_i - \mathbf{c}_{t_i}\|)]$$

where N is the number of examples, \mathbf{x}_i is the embedding of the i^{th} example, \mathbf{c}_{t_i} is the centroid of the example's ground truth class and δ_i is defined as:

$$\delta_i = \begin{cases} 1 & \text{if } \|\mathbf{x}_i - \mathbf{c}_{t_i}\| > r_{t_i}, \\ 0 & \text{if } \|\mathbf{x}_i - \mathbf{c}_{t_i}\| \leq r_{t_i} \end{cases}$$

As described earlier, examples whose embeddings do not belong to the spherical boundary of the closest centroid are considered OOD, while the others are classified as the centroid's class k :

$$y = \begin{cases} k & \text{if } \|\mathbf{x} - \mathbf{c}_k\| \leq r_k, \\ \text{out-of-domain} & \text{otherwise.} \end{cases}$$

subject to:

$$k = \arg \min_{k \in K} \|\mathbf{x} - \mathbf{c}_k\|$$

where y is the resulting label and K is the set of all in-domain intents.

6.5 Proposed Method

Our Proposed Method is inspired by ADB (6.4) – it is also a two-step algorithm based on metric learning and creation of decision boundaries specific to each class. However, we experiment with LMCL and Triplet Loss metric learning and we find the best radius, in terms of compromise between the distances from the radius to either the ID or OOD examples, using an iterative method with a stopping criterion.

After either LMCL or Triplet Loss metric learning, we work in the One-vs-Rest setting [23] – we treat examples from one in-domain intent as positive, while other intents with their respective examples are considered out-of-domain. This is another difference from ADB, where the examples are regarded as OOD dynamically, based on the current radius during training.

Thus, for every in-domain class k , we iteratively increase the possible value of radius r_k , evaluate our criterion and stop iterating when it returns a negative value.

The stopping criterion is defined as:

$$criterion = \frac{\sum_{\mathbf{x} \in OOD} (\|\mathbf{x} - \mathbf{c}_k\| - r_k)}{N_{OOD}} - \frac{\sum_{\mathbf{x} \in ID} (r_k - \|\mathbf{x} - \mathbf{c}_k\|)}{N_{ID}} \frac{N_{OOD}}{N_{ID}} \frac{1}{\alpha}$$

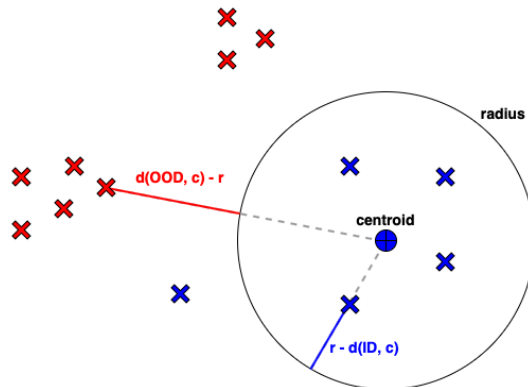


Figure 6.1. Stopping criterion of Our Proposed Method. ID is an in-domain example, OOD is an out-of-domain example, c is the centroid, d denotes the distance between two objects and r is the radius.

where N_{OOD} (respectively N_{ID}) is the number of out-of-domain (resp. in-domain) examples, \mathbf{x} is the embedding of an example, k is the current class, \mathbf{c}_k is the centroid of class k as described in ADB, r_k is the current radius for class k and α is a hyperparameter. The stopping criterion is shown in Figure 6.1.

We can observe that the criterion returns a negative value when the in-domain examples outweigh the out-of-domain. Additionally, we multiply the weight of in-domain examples by $\frac{N_{OOD}}{N_{ID}}$ in order to counter the imbalance between the number of OOD and ID examples, and by the hyperparameter α which normalizes the importance of OOD detection in opposite to ID classification. We have empirically observed that lower values of α are better for lower numbers of in-domain classes.

Finally, classification is performed as in ADB – if constrained by its decision boundary, the example is classified as the closest centroid’s class k , otherwise it is considered out-of-domain:

$$y = \begin{cases} k & \text{if } \|\mathbf{x} - \mathbf{c}_k\| \leq r_k, \\ \text{out-of-domain} & \text{otherwise.} \end{cases}$$

subject to:

$$k = \arg \min_{k \in K} \|\mathbf{x} - \mathbf{c}_k\|$$

where y is the resulting label and K is the set of all in-domain intents. A visualization of our method on toy data is available in Figure 6.2.

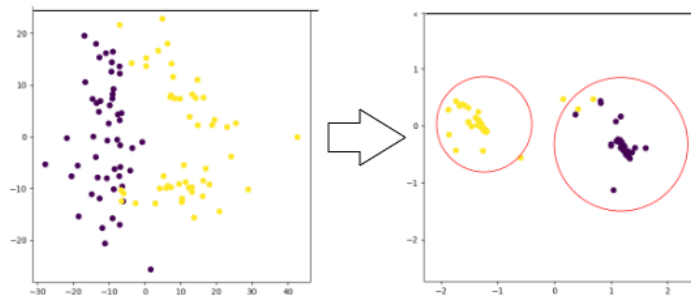


Figure 6.2. Effect of metric learning and Our Proposed Method on toy data.

Chapter 7

Experiments

This chapter includes the evaluations that were performed on the various datasets. We also describe the used metrics and provide further details about our experimental setup.

7.1 Metrics

We evaluate the results of the employed methods on several performance criteria – accuracy over the predefined in-domain intents, recall on out-of-domain queries and false rejection rate (FRR).

We also take into account the models' time requirements for training and inference, as well as their memory consumption. Both are very important, since large requirements of either time or RAM would compromise the usability of the models in real-world applications, where these parameters are essential.

7.1.1 Precision and Recall

For in-domain intents, we are interested in accuracy – the fraction of correct predictions among all predictions that have an ID ground truth label. On the other hand, recall – the fraction of relevant predictions – gives more weight to out-of-domain intents. Measuring recall provides a needed additional insight on OOD intents, because if the model misclassified an OOD user input, the system would give an unrelated response.

7.1.2 FRR

FRR is the ratio of falsely rejected in-domain sentences to all ID sentences. By minimizing FRR, the error of considering ID sentences as OOD is reduced.

FRR minimization is critical to conversational AI. Since chatbots use the conversational-centric style of dialogues, frequent in-domain misclassifications would prevent the system from guiding the user through the predefined conversation flow.

7.1.3 Formulae

Recall and FRR can be expressed by the following formulae:

$$\text{recall} = \frac{TP}{TP + FN}$$
$$\text{FRR} = \frac{FP}{FP + TN}$$

where

- TP – predicted as out-of-domain and true label is out-of-domain
- TN – predicted as in-domain and true label is in-domain
- FP – predicted as out-of-domain and true label is in-domain
- FN – predicted as in-domain and true label is out-of-domain

7.2 Experimental Setup

The neural networks were implemented in TensorFlow 2 [24]. The classification models were trained for 40 epochs and optimized using Adam [25], with its default learning rate $1e-3$. The NNs used for metric learning were trained for 20 epochs and Adam’s learning rate was set to $1e-4$.

LMCL’s two hyperparameters were set as suggested in the original paper: $m = 0.35$ and $s = 64$.

The default margin hyperparameter used in Triplet Loss’s TensorFlow implementation was left unchanged: $\alpha = 1$.

Local Outlier Factor was again used with its default scikit-learn [26] implementation.

In our proposed method, we used 3 different hyperparameter values, depending on the dataset:

- CLINC150: $\alpha = 1.45$
- ROSTD: $\alpha = 0.2$
- Supervisor Dataset: $\alpha = 1.0$

Both training and inference times include creating sentence embeddings from the respective training and test splits. Training time includes training the model. Inference time includes prediction of all testing examples. Memory usage, calculated using the Python library psutil¹, includes the size of the model and of the training embeddings.

All the computations were executed on a dual-core *Intel(R) Core(TM) i5-7267U CPU @ 3.10GHz*.

7.3 ADB Performance

When evaluating the performance of ADB on the CLINC150 dataset, we have found significantly worse results (see Table 7.1) than claimed in the original paper.

Method	Accuracy	Recall	FRR
ADB	38.4 / 38.7 / 37.0	99.5 / 99.8 / 99.4	60.8 / 60.9 / 62.2

Table 7.1. ADB results – USE-DAN / USE-TRAN / SBERT embeddings.

We suspect this might be caused by some differences in implementation, especially in training, which were not mentioned in the paper.

However, we have found that we can boost ADB’s performance by replacing Softmax Loss metric learning with LMCL or Triplet Loss, and by adding 3 hidden layers with the ReLu activation function to the model. These layers perform further metric learning based on the model’s Boundary Loss. The results (see Table 7.2) then improve.

Method	Accuracy	Recall	FRR
ADB _{LMCL}	92.9 / 95.4 / 89.9	75.4 / 75.2 / 75.8	3.9 / 2.1 / 5.9
ADB _{Triplet}	91.2 / 95.0 / 89.1	77.2 / 74.3 / 73.8	4.6 / 2.1 / 5.8

Table 7.2. New ADB results – USE-DAN / USE-TRAN / SBERT embeddings.

Therefore, we will only use our new version of ADB in future evaluations.

¹ <https://github.com/giampaolo/psutil>

7.4 CLINC150

As we can observe in Table 7.3, the best results were achieved with USE-TRAN embeddings. As for accuracy, the best method was BaselineNN. On the other hand, even though BaselineNN received out-of-domain training data, its recall is significantly lower than the recall of ADB or Our Proposed Method.

Altogether, Our Proposed Method with LMCL metric learning achieves the best results. It has a high accuracy, the best recall and FRR is reasonably low.

Method	Accuracy	Recall	FRR
Cosine Similarity	87.7 / 92.1 / 83.3	12.8 / 14.6 / 6.1	0.0 / 0.0 / 0.1
BaselineNN	95.1 / 96.5 / 92.0	41.1 / 51.4 / 29.8	0.2 / 0.1 / 0.3
DeepUnk	90.3 / 92.7 / 85.9	39.6 / 38.4 / 54.3	4.5 / 4.3 / 8.5
ADB _{LMCL}	92.9 / 95.4 / 89.9	75.4 / 75.2 / 75.8	3.9 / 2.1 / 5.9
ADB _{Triplet}	91.2 / 95.0 / 89.1	77.2 / 74.3 / 73.8	4.6 / 2.1 / 5.8
Our_{LMCL}	92.2 / 94.6 / 89.6	81.3 / 83.9 / 79.3	4.9 / 3.1 / 6.8
Our _{Triplet}	92.5 / 94.8 / 89.6	72.5 / 78.6 / 69.7	2.6 / 2.1 / 4.8

Table 7.3. Results on CLINC150 – USE-DAN / USE-TRAN / SBERT embeddings.

As shown in Table 7.4, USE-DAN’s worse performance in comparison with USE-TRAN is compensated by its speed and lower memory requirements. SBERT has, considering its low accuracy and recall results, surprisingly high requirements. Compared to other methods that rely on a neural network, Our Proposed Method’s training is significantly faster.

Method	Training (s)	Inference (s)	Memory (MB)
Cosine Similarity	0.9 / 65.4 / 473.1	2.2 / 23.9 / 160.6	979.7 / 1277.9 / 767.5
BaselineNN	23.6 / 94.9 / 503.2	1.5 / 21.9 / 146.3	917.1 / 1140.3 / 924.7
DeepUnk	70.2 / 141.8 / 614.4	16.5 / 40.4 / 178.4	738.1 / 440.8 / 948.3
ADB _{LMCL}	64.1 / 134.0 / 643.7	10.6 / 30.1 / 224.4	664.4 / 1077.0 / 886.6
ADB _{Triplet}	61.6 / 139.9 / 647.8	8.0 / 34.4 / 221.5	672.9 / 1081.2 / 894.0
Our _{LMCL}	9.7 / 77.2 / 544.6	15.1 / 35.1 / 245.0	649.2 / 1056.8 / 850.5
Our _{Triplet}	9.3 / 111.9 / 528.3	9.2 / 40.5 / 220.8	661.7 / 788.0 / 862.3

Table 7.4. Time and memory on CLINC150 – USE-DAN / USE-TRAN / SBERT embeddings.

In Table 7.5, we can see that LMCL metric learning is significantly faster than Triplet Loss.

Type	Metric learning (s)
LMCL	44.4 / 110.9 / 507.0
Triplet Loss	389.0 / 460.6 / 825.1

Table 7.5. Metric learning time on CLINC150 – USE-DAN / USE-TRAN / SBERT embeddings.

7.5 ROSTD

ROSTD results (see Table 7.6) continue to demonstrate the strength of USE-TRAN over USE-DAN. The metrics of all methods, excluding ADB with LMCL metric learning, benefitted from using USE-TRAN. We are not able to explain the abnormal results of ADB with LMCL metric learning.

As for accuracy, all methods gave excellent results. In comparison with CLINC150, out-of-domain recall of all methods, except ADB, substantially improved. As shown later in Section 7.7, ADB’s performance is penalized by ROSTD’s low number of classes.

Although BaselineNN has better overall results, we highlight Our Proposed Method with LMCL metric learning, as it yields comparatively strong results in all metrics without the need of OOD training data.

Method	Accuracy	Recall	FRR
Cosine Similarity	97.9 / 98.1 / 96.9	62.6 / 69.4 / 58.7	0.0 / 0.0 / 0.1
BaselineNN	98.8 / 98.9 / 98.6	93.2 / 96.2 / 89.8	0.1 / 0.0 / 0.1
DeepUnk	90.9 / 89.2 / 83.9	80.7 / 81.1 / 78.3	8.4 / 10.2 / 15.4
ADB _{LMCL}	98.6 / 98.8 / 98.3	30.7 / 22.7 / 17.8	0.5 / 0.4 / 0.6
ADB _{Triplet}	96.1 / 97.1 / 97.2	39.5 / 52.2 / 27.5	2.1 / 1.2 / 1.1
Our_{LMCL}	95.8 / 96.4 / 95.6	92.8 / 95.8 / 78.5	3.9 / 3.4 / 4.0
Our _{Triplet}	92.8 / 94.1 / 91.9	93.8 / 98.0 / 84.6	6.8 / 5.5 / 7.6

Table 7.6. Results on ROSTD – USE-DAN / USE-TRAN / SBERT embeddings.

As we can observe in Table 7.7, the time requirements are consistent with the results on CLINC150. Conversely, USE-DAN is more memory demanding than USE-TRAN. We believe this is caused by the fact that ROSTD contains shorter sentences than CLINC150, penalizing USE-DAN’s constant memory usage and rewarding USE-TRAN’s space complexity dependent on sentence length.

Method	Training (s)	Inference (s)	Memory (MB)
Cosine Similarity	2.2 / 171.0 / 810.5	6.5 / 80.8 / 316.0	671.4 / 263.7 / 843.1
BaselineNN	34.8 / 253.9 / 966.8	3.1 / 88.1 / 315.1	694.5 / 230.6 / 861.3
DeepUnk	187.7 / 277.3 / 1220.6	53.1 / 114.5 / 404.9	886.9 / 859.9 / 1162.0
ADB _{LMCL}	130.3 / 253.9 / 1230.2	4.3 / 51.7 / 410.4	765.1 / 597.6 / 976.5
ADB _{Triplet}	133.9 / 248.5 / 1233.8	5.0 / 52.0 / 412.9	771.6 / 398.9 / 902.1
Our _{LMCL}	5.6 / 112.7 / 1004.5	4.4 / 46.2 / 447.5	745.4 / 386.3 / 961.3
Our _{Triplet}	5.6 / 116.5 / 1021.7	4.4 / 46.7 / 423.4	765.9 / 470.9 / 976.5

Table 7.7. Time and memory on ROSTD – USE-DAN / USE-TRAN / SBERT embeddings.

The results in Table 7.8 confirm the previous findings on LMCL’s efficiency.

Type	Metric learning (s)
LMCL	78.8 / 206.5 / 925.2
Triplet Loss	756.0 / 831.3 / 1581.6

Table 7.8. Metric learning time on ROSTD – USE-DAN / USE-TRAN / SBERT embeddings.

7.6 Supervisor Dataset

Considering the results on the two previous datasets, we have decided to not include evaluations with SBERT embeddings or Triplet Loss metric learning.

On Supervisor Dataset, DeepUnk achieves the best accuracy at the expense of recall results. Despite its simplicity, Cosine Similarity has surprisingly good results.

Overall, Our Proposed Method with LMCL metric learning achieves very high accuracy and the best recall. These outstanding results combined with the lack of need of out-of-domain training data make of Our the best method.

Method	Accuracy	Recall	FRR
Cosine Similarity	96.1 / 97.8	88.7 / 89.3	0.0 / 0.0
BaselineNN	93.2 / 95.6	93.1 / 94.7	0.2 / 0.3
DeepUnk	97.8 / 98.2	11.4 / 12.0	0.4 / 0.6
ADB _{LMCL}	91.6 / 94.3	75.5 / 76.0	7.2 / 5.1
Our_{LMCL}	94.8 / 96.2	97.8 / 98.1	4.3 / 3.3

Table 7.9. Results on Supervisor Dataset – USE-DAN / USE-TRAN embeddings.

As shown in Table 7.10 and Table 7.11, the time and memory requirements of the employed methods are generally in agreement with previous measurements. We can observe that due to their simplicity, the combination of Cosine Similarity with USE-DAN embeddings is the most efficient method by a large margin, since it was not as penalized as the other methods by the dialogue structure of Supervisor Dataset.

Method	Training (s)	Inference (s)	Memory (MB)
Cosine Similarity	3.7 / 1849.9	36.9 / 627.5	590.5 / 1003.9
BaselineNN	2088.8 / 4396.5	146.5 / 707.2	1243.6 / 1190.4
DeepUnk	3456.5 / 4955.9	365.2 / 883.6	1505.6 / 1788.6
ADB _{LMCL}	3539.4 / 6464.9	133.4 / 1072.8	2474.1 / 2555.9
Our _{LMCL}	701.7 / 2947.3	142.0 / 911.4	764.6 / 1304.8

Table 7.10. Time and memory on Supervisor Dataset – USE-DAN / USE-TRAN embeddings.

Type	Metric learning (s)
LMCL	2335.9 / 3720.4

Table 7.11. Metric learning time on Supervisor Dataset – USE-DAN / USE-TRAN embeddings.

7.7 Limited Sentences and Intents

In a real-world setting, conversational agents often use fewer intents with a limited number of training sentences. Therefore, we compare the behaviour of Our Proposed Method and ADB when there is a limited number of in-domain intents and training

sentences. We have chosen these two methods because they both do not need out-of-domain training data and because of their similar approach to the problem. USE-TRAN embeddings with LMCL metric learning and the CLINC150 dataset are employed.

For each comparison, we select 10, 25 or 50 classes and limit the number of training sentences to 4, 8, \dots , 100. The used classes and sentences are chosen randomly. The results are then computed as the average over 5 runs.

As we can observe in Figure 7.1, Figure 7.2 and Figure 7.3, both methods achieve very high accuracy rates. However, Our Proposed Method is more appropriate for a setting with a low number of classes, as it needs less training sentences than ADB.

As for out-of-domain recall, the difference between the two methods is more significant. Regardless of the number of intents, ADB is not able to achieve the same recall as Our Proposed Method.

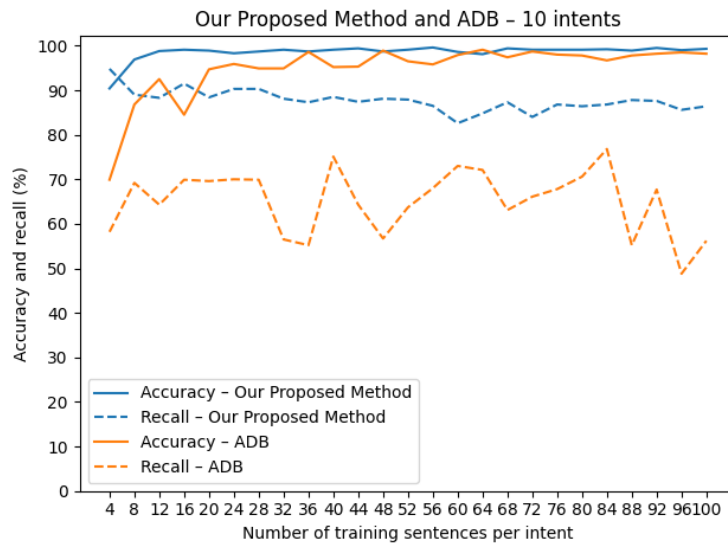


Figure 7.1. Comparison between Our Proposed Method and ADB on 10 intents.

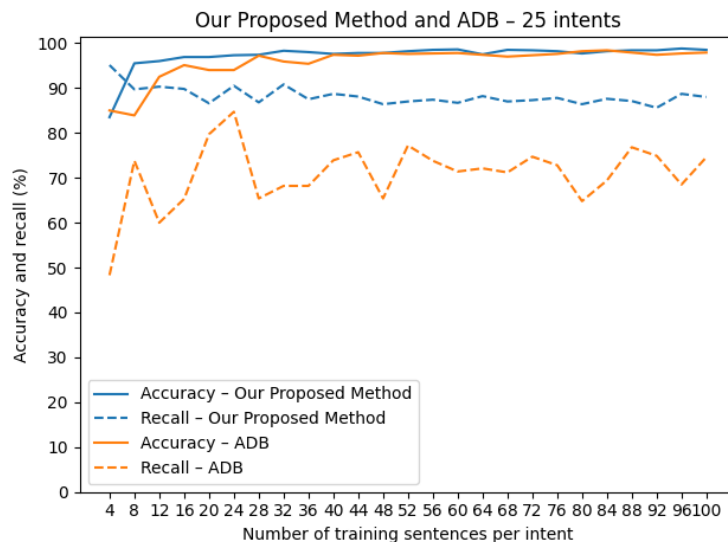


Figure 7.2. Comparison between Our Proposed Method and ADB on 25 intents.

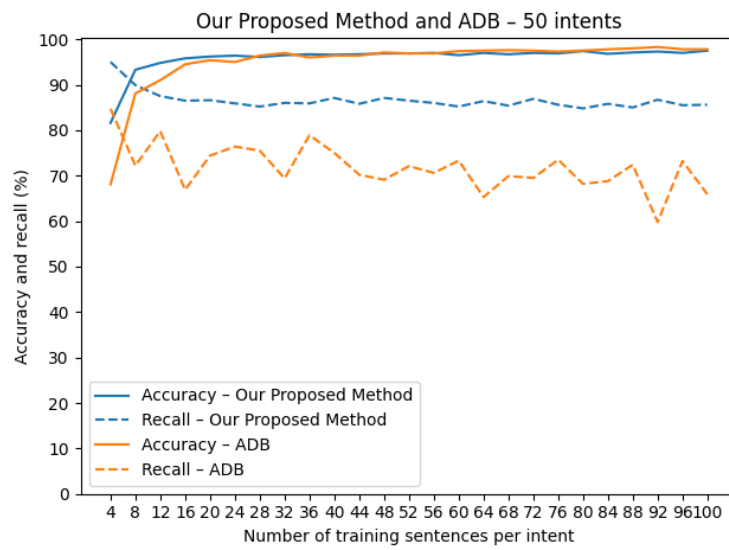



Figure 7.3. Comparison between Our Proposed Method and ADB on 50 intents.



Chapter 8

Conclusion

In this thesis, we have introduced the problem of out-of-domain anomaly detection and reviewed the current methods employed for its solution. We have also explained the core concepts behind neural networks and the role of loss functions in model training, discussing the positive impact on embeddings brought by metric learning.

Subsequently, we have proposed a two-step algorithm based on the combination of metric learning and creation of boundaries using a novel stopping criterion. Since out-of-domain data are difficult to collect, one of the strong points of Our Proposed Method is its lack of need of OOD training data.

Several experiments on a variety of datasets and embeddings were performed. These evaluations have confirmed the strength of Our Proposed Method which has, with feasible computational requirements, significantly outperformed the current state-of-the-art.

We believe that the outcomes of this thesis will have positive implications for building better conversational agents.

References

- [1] Robert J. Moore, and Raphael Arar. *Conversational UX Design: An Introduction*. In: Robert J. Moore, Margaret H. Szymanski, Raphael Arar, and Guang-Jie Ren, eds. *Studies in Conversational UX Design*. Cham: Springer International Publishing, 2018. 1–16. ISBN 978-3-319-95579-7.
https://doi.org/10.1007/978-3-319-95579-7_1.
- [2] Sarah E. Finch, James D. Finch, Ali Ahmadvand, Ingyu, Choi, Xiangjue Dong, Ruixiang Qi, Harshita Sahijwani, Sergey Volokhin, Zihan Wang, Zihao Wang, and Jinho D. Choi. *Emora: An Inquisitive Social Chatbot Who Cares For You*. 2020.
- [3] Jan Pichl, Petr Marek, Jakub Konrád, Petr Lorenc, Van Duy Ta, and Jan Šedivý. *Alquist 3.0: Alexa Prize Bot Using Conversational Knowledge Graph*. 2020.
- [4] C. Maria Keet. *Open World Assumption*. In: Werner Dubitzky, Olaf Wolkenhauer, Kwang-Hyun Cho, and Hiroki Yokota, eds. *Encyclopedia of Systems Biology*. New York, NY: Springer New York, 2013. 1567–1567. ISBN 978-1-4419-9863-7.
https://doi.org/10.1007/978-1-4419-9863-7_734.
- [5] Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. *An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction*. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019.
<https://www.aclweb.org/anthology/D19-1131>.
- [6] Varun Gangal, Abhinav Arora, Arash Einolghozati, and Sonal Gupta. Likelihood Ratios and Generative Classifiers for Unsupervised Out-of-Domain Detection In Task Oriented Dialog. *arXiv preprint arXiv:1912.12800*. 2019,
- [7] Sebastian Schuster, Sonal Gupta, Rushin Shah, and Mike Lewis. *Cross-Lingual Transfer Learning for Multilingual Task Oriented Dialog*. 2019.
- [8] Prof. Loc Vu-Quoc. *Neuron3*. 2018.
<https://commons.wikimedia.org/wiki/File:Neuron3.png>.
- [9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*. 1989, 2 (5), 359-366. DOI [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8).
- [10] IBM Cloud Education. *Deep neural network*. 2020.
https://1.cms.s81c.com/sites/default/files/2021-01-06/ICLH_Diagram_Batch_01_03-DeepNeuralNetwork-WHITEBG.png.
- [11] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. *CosFace: Large Margin Cosine Loss for Deep Face Recognition*. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018. 5265-5274.

- [12] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, DOI 10.1109/cvpr.2015.7298682.
- [13] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. *Universal Sentence Encoder*. 2018.
- [14] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. *A large annotated corpus for learning natural language inference*. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, 2015. 632–642. <https://www.aclweb.org/anthology/D15-1075>.
- [15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention is All You Need*. In: 2017. <https://arxiv.org/pdf/1706.03762.pdf>.
- [16] Nils Reimers, and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019.
- [17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019.
- [18] Matthew Schultz, and Thorsten Joachims. *Learning a Distance Metric from Relative Comparisons*. In: S. Thrun, L. Saul, and B. Scholkopf, eds. *Advances in Neural Information Processing Systems*. MIT Press, 2004. <https://proceedings.neurips.cc/paper/2003/file/d3b1fb02964aa64e257f9f26a31f72cf-Paper.pdf>.
- [19] Ting-En Lin, and Hua Xu. *Deep Unknown Intent Detection with Margin Loss*. 2019.
- [20] Markus Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. *LOF: Identifying Density-Based Local Outliers*. In: *PROCEEDINGS OF THE 2000 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 2000. 93–104.
- [21] Hanlei Zhang, Hua Xu, and Ting-En Lin. *Deep Open Intent Classification with Adaptive Decision Boundary*. 2021.
- [22] Makarand Tapaswi, Marc T. Law, and Sanja Fidler. *Video Face Clustering with Unknown Number of Clusters*. 2019.
- [23] Ryan Rifkin, and Aldebaro Klautau. In Defense of One-Vs-All Classification. *J. Mach. Learn. Res.*. 2004, 5 101–141.
- [24] Martin 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 Mane, 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 Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. <https://www.tensorflow.org/>. Software available from tensorflow.org.

-
- [25] Diederik P. Kingma, and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 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*. 2011, 12 2825–2830.