

# Using Optimal Embeddings to Learn New Intents with Few Examples: An Application in the Insurance Domain

Shailesh Acharya\*

American Family Insurance, Machine Learning Research  
Group  
sachary1@amfam.com

Glenn Fung

American Family Insurance, Machine Learning Research  
Group  
gfung@amfam.com

## ABSTRACT

The ubiquitous adoption of Conversational Agents (CA) in commercial settings is changing the way industries interact with their customers. Intent classification is an important first step in designing an efficient CA. Every intent that the CA can recognize is represented by a set of natural language examples that are used by the system to learn how to map any user's utterance to the corresponding intent. However, when a new intent is introduced, there are usually not enough examples to train the intent appropriately. In this paper we propose a hybrid system that combines a traditional Deep Neural Network-based classification approach with few-shot learning strategies. The simple but yet effective proposed approach achieves good performance for newly introduced intents with few training examples while maintaining performance for previously known intents. We show the potential of the proposed approach on a data generated by a deployed chat system for the insurance domain. To demonstrate that the propose approach can generalize to other domains, we also perform experiments in a publicly available dataset where we obtain similar approach-substantiating results.

## CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**:  
*Learning latent representations.*

## KEYWORDS

Dataset, Fewshot learning, Embedding, Chatbot, Intent Classification

## 1 INTRODUCTION

Conversational Agent and chatbots are getting increasingly popular in the industry that frequently interact with customers. A successful conversational agent can alleviate the burden on customer representatives by understanding the customer's query (often presented in natural language) and guiding the user towards a solution. Intent classification is an important first step in designing an intelligent chatbot. It allows chatbot to understand the intent of customer and drive the conversation. For example, if a customer of an insurance company asks *What is the minimum liability*

*limit for state of Wisconsin?* then the general intent or superclass that generalizes this query or utterance can be something like *AUTO\_COVERAGE\_LIABILITY\_LIMIT*. The chatbot should be able to identify the intent and get the right answer from the knowledge base.

These intents are defined by business and added/modified according to business needs. Every time a new product/service is launched, new intents relating to that service are added to the chatbot. For instance; if a new discount scheme is introduced for usage-based insurance (UBI) then an associated intent for the chatbot can be *UBI\_ELIGIBILITY*. There could be more than one intent associated with the new service/offering depending on the scope or complexity of it. Often, the newly added intents could be of significant importance to the business for two reasons; first, it could be associated to a newly introduced "hot" popular service or offering and hence much more likely to be queried or solicited by customers and second, there could be a greater business drive to upsell this newly introduced service. For these reasons, it is very important for the chatbot to do well in the intent category/categories relating to this service.

An underlying problem with the newly added intents is the lack of enough training examples to train an accurate intent classifier since there is no past interaction with customers regarding this topic. Normally, a subject matter expert comes up with different ways of asking questions about that service in order to gather several training samples for that intent class. Such a collection of training examples is limited in number and variation. This can present some challenges to the intent prediction classifier; in spite of having few training examples to begin with, the likelihood of getting queries related to this intent could be high. Furthermore, the cost of misclassifying examples belonging to this new intent class could be potentially much higher to the business.

Intent classification is usually formulated as a multiclass (sometimes multilabel) classification problem and the use of deep-neural-network-based classifiers is very popular. A well-known trait of training deep neural networks (NN) classifiers is that they need large amounts of labeled training data to provide satisfactory classification performance. They generally perform poorly on categories with fewer training examples. Hence, the ability of deep NN to extract complex statistics and learn high level features from vast datasets is proven. However, most current deep learning approaches have poor sample efficiency in contrast to human perception - even a child could recognise a bird after previously seeing a few bird pictures.

There are abundant recent works in few-shot learning [7, 8]: an area of machine learning dedicated to solving problems with very few examples per class (but large number of class labels).

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

*KDD Converse'20, August 2020,*

© 2020 Copyright held by the owner/author(s).

The main idea behind few-shot learning is to efficiently combine meta-learning: that uses prior knowledge to define learning strategies and metric learning: that aims to learn semantic embeddings using a distance loss function and data augmentation (by synthesizing data) to facilitate the learning process using fewer labeled examples. Many few shot learning models such as Prototypical Networks [8], Facenet [7] and Matching Network [10] used optimized embedding transformations and distance-based learning to embed examples belonging to the same class close to each other in a new generated embedding space. This approach has proven to be more effective than a regular fully connected NN classifier if the dataset has large number of classes but fewer examples per class. Few shot approaches have also been successfully applied to intent classification domain [1] in the past.

The new intent classification problem described above is an interesting combination of traditional learning and few-shot learning since we have a large number of intent classes with sufficiently large number of examples collected from past interaction with customers but we also have the newly added intents with very few examples. Recent work aims to address the problem of learning new intents with few examples. In [5] the author's propose using a combination of data augmentation combined with few-shot learning with modest results, while in [4] only the effect of few-shot learning for the same task is studied. However, in contrast with our work, these two recent papers don't study how to incorporate the new learned intents with the existing intent classification system as described next.

These ideas inspire a simple but effective approach: to combine the strength and low data dependence of distance-based learning with a regular industry standard NN classifier trained on classes (intents) for which we have enough data, to build an ensemble classifier that improves performance on new recently added intent categories while maintaining the performance on the already existing ones. This approach plays with with current deployed CA implementations making it very practical in most industrial settings.

## 2 PRELIMINARIES

### 2.1 Triplet Network

Facenet introduced an image embedding model that encodes face images as vectors in a Euclidean space where distances directly correspond to facial similarities [7]. The model consists of a weight-sharing triplet network trained to minimize a loss function based on large margin nearest neighbours [11]. A triplet  $i$  consists of two examples  $x_i^{p_1}, x_i^{p_2}$  from the same class known as "positive"  $x_i^{p_1}$  and anchor  $x_i^{p_2}$  and one example  $x_i^n$  from a different class known as "negative". Triplet loss minimizes distance between the positive points while maximizing the distance between the anchor  $x_i^{p_2}$  and the negative example. This idea is illustrated in Figure 1. The objective function to minimize is:

$$\sum_i^N \|f(x_i^{p_1}) - f(x_i^{p_2})\|_2^2 - \|f(x_i^{p_1}) - f(x_i^n)\|_2^2 + \alpha \quad (1)$$

where  $\alpha$  is a margin enforced between positive and negative pairs and  $N$  is the number of triplets in training set.



Figure 1: Triplet loss forces anchor and positive to come closer while pushing negative away

One key advantage of this approach is its ability to learn from very few examples per class given that there are sufficiently large number of classes. Instead of classifying images to different categories, the Facenet model learns the notion of similarity between faces. For the downstream tasks, the encoder model can be used to encode images into embedding vectors for tasks such as clustering, nearest-neighbor-based classification, ranking, etc. The general approach used to train Facenet can be easily applied in any other domain including text. For the intent classification problem, we can instantiate our problem as a triplet network formulation. Instead of the encoder used in Facenet, we include our own text encoder by adding fully connected layers on top of the Universal Sentence Encoder [2]. The loss formulation in equation 2 and the training procedure we used, however, is akin to the ones described in [7]. Our proposed model learns a transformation that maps sentences belonging to same intents close to one another in the euclidean space and far apart from another sentence belonging to different intent classes.

### 2.2 Prototypical Network

Prototypical networks learn a non-linear mapping of the input feature space into an embedding space and create class's prototype  $c_k \in R^M$  to be the mean of the embedded support points belonging to the class  $k$ .

$$c_k = 1/|S_k| \sum_{(x_i, y_i) \in S_k} f_\phi(x_i) \quad (2)$$

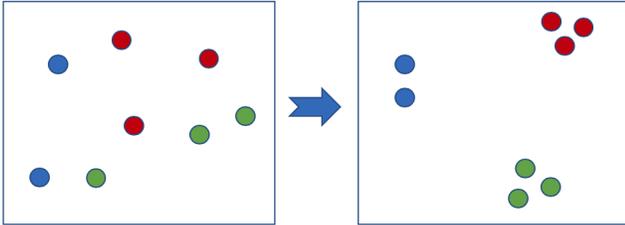
where  $S_k = \{(x_1^k, y_1^k), (x_2^k, y_2^k) \dots (x_n^k, y_n^k)\}$  is the set of support points belonging to class  $k$  and  $f_\phi : R^D \rightarrow R^M$  is a non linear transformation with trainable parameters. For a query point  $x$ , prototypical networks produce a distribution over classes based on a softmax over distances to the prototypes in the embedding space:

$$p_\phi(y = k|x) = \frac{\exp(-d(f_\phi(x), c_k))}{\sum_j \exp(-d(f_\phi(x), c_j))} \quad (3)$$

The corresponding loss function is then derived as the negative log-probability  $-\log p_\phi(y = k|x)$  of true class  $k$ .

Similar to triplet networks, we can interpret prototypical networks as a combination of a encoder model and a loss formulation that ensures similar examples cluster in a tight formation. Thus, we can then use a similar approach to frame our intent classification problem to fit in the prototypical network framework. In order to do this, we replace the encoder with our text encoder(USE followed by two FC layers) and follow the training procedure identical to

the one described in [8]. However, we needed to make some modifications to the inference procedure to fit to our specific use-case. We describe our modified inference algorithm in more detail in the next section.



**Figure 2: Both triplet and prototypical networks map input to a vector space where examples from same class form tight clusters**

The key aspect of both triplet network and prototypical network is the encoder function  $f : \theta_1 \rightarrow \theta_2$  that transforms input from one vector space ( $\theta_1$ ) to another ( $\theta_2$ ) based on some distance based loss minimization. As a result, each class forms a tight cluster separated from one another in this new vector space as seen in figure 2. One key strength of this approach compared to regular Deep Neural Network (DNN) multiclass classifier is its ability to learn from very few examples per class given that there are sufficiently large number of classes. DNN classifiers while generally performing good on classification tasks with big datasets, performs poorly on categories with few examples. This is the behaviour we observe in our experiments as well. In general, intent classification datasets are highly unbalanced with some intents having large number of examples while some newly added intents having very few examples. In order to combine the strength of distance-based learning (Triplet or prototypical network) and the practicality and accuracy of regular deep NN multiclass classifiers we propose a hybrid approach that combines both models to build a classifier that is more robust to less frequent intent classes.

### 3 DATASETS

#### 3.1 Amfam Chatbot Intent Dataset

The Amfam chatbot intent dataset (ACID) contains 174 unique intents related to topics driven by customers contacting an insurance company. Each intent represents a particular course of action for the chatbot. For example; if a customer says *I got a new cell number and need to update my account info* then it belongs to the intent class INFO\_UPDATE\_PHONE\_NUM. The intent prediction dataset was collected from past interaction of customers with our service representatives at American Family Insurance. Subject matter experts created different intent categories and hand-picked examples belonging to each intent category. The dataset contains 175 unique intents. It is split into a training set that contains a total of 11,130 examples and test set that contains total of 11,042 examples. The distribution across the intent classes is highly skewed with the smallest intent class containing 10 examples and the biggest class containing 378 examples in the training set. The

dataset will be released to public and can be downloaded from: <https://github.com/AmFamMLTeam/ACID>

#### 3.2 CLINC150 dataset

The CLINC150([6]) is another publicly available intent classification dataset that contains large collection of in-scope and out-of-scope intents and examples from different domains. The dataset has several variants; Full, Small, Imbalanced and OOS+. Depending on the variant, it has different train/test split. For our experiment, we take the FULL version which contains 150 "in-scope" intent classes, each with 100 examples in train, 30 examples in test and 20 in validation. The dataset also contains out of scope examples ("out-of-domain" or "out-of-distribution") which we do not include in our experiments. The dataset contains intents from different domains related to travel, expenses, reminders, fees, and several intents associated with smart assistant such as joke, play\_music, etc.

## 4 MODEL AND ALGORITHM

### 4.1 Model

We used a pretrained Universal Sentence Encoder (USE) [2] as a base encoder. USE projects input text into a 512 dimensional vector space. We treat this base encoder as a preprocessing step and all subsequent models use these sentence embeddings as their input.

We created a baseline model by adding two fully connected layers to the output from USE. In the last layer of this network we incorporated a softmax activation function to produce a probability distribution over the intent classes.

Similarly, we also created both a triplet and a prototypical network by adding two fully connected layers to the output from the USE network. We followed the same training process described in the original paper for these two networks. During inference, however, we use a different strategy. This is because we are only interested in the final embedding vectors these networks produce which we use to perform our own distance-based classification. The choice of the number of layers, the dimension of the layers and other network parameters for all the three networks was determined by hyperparameter search. Details of those hyperparameters are provided in appendix section.

It is important to note that our goal is not to compare the baseline model against the triplet network or the prototypical network. Instead, our goal is to compare the performance of the baseline model against an ensemble model (baseline combined with triplet/prototypical) on the newly added intent classes. We use the ensemble model during inference by combining predictions from the baseline model and the corresponding triplet/prototypical network. The underlying idea of this ensemble is very simple; we use the embedding model (triplet/prototypical) to determine if the incoming testing example belongs to the newly added intent class, if not, we pass it through the baseline classifier to get the corresponding class label. We will describe the inference strategy in more detail in algorithm 1 and algorithm 2. For simplicity, we will use the term "embedding model" to refer to the triplet/prototypical networks in the following sections.

## 4.2 Algorithm

**4.2.1 Adding a new intent.** To simulate a scenario where a new intent class is added with few available examples, we randomly select one intent class  $c_k$  from dataset and retain only 13 examples (randomly chosen) for that class in the training set. The number 13 is chosen considering the minimum examples required to train the prototypical network with the settings we used. For ACID dataset, we choose  $c_k$  from a subset of intent classes with at least 80 examples because ACID dataset is highly unbalanced. This guarantees that the test set is big enough to have diverse distribution of samples and correctly simulates real world scenario where new intent categories have few examples to begin with but during inference queries coming from customers is much more diverse. Inference is done in two stages; first, we use the embedding model to decide if an example belongs to the selected intent class  $c_k$ . This is done by comparing the average distance between the test example and all training examples belonging to class  $c_k$  with a pre-determined threshold. If average euclidean distance is greater than the pre-determined threshold, we pass the example through the baseline NN classifier and get the class label. We use the following algorithm during inference to score a new example:

---

### Algorithm 1 Inference Single:

---

- 1: Calculate threshold  $\{\theta\}$
  - 2: Transform input  $x$  to embedding vector,  $v = f\phi(x)$
  - 3: Calculate distance between  $v$  and all training set examples belonging to intent class  $c_k$ ;  $d_k = \text{dist}(v, v_{train}), v_{train}$  in class  $c_k$  followed by average distance  $d_k = \text{average}(d_k)$
  - 4: If  $d_k < \theta$ , assign example to class  $c_k$
  - 5: If  $d_k > \theta$ , pass the example through baseline classifier to get the class label
- 

**4.2.2 Threshold.** The distance threshold  $\theta$  is an important parameter calculated based on the intra-class distance matrix for the intent class  $c_k$  in both the training and the validation set. We choose 10 equally spaced points between median and max of the distance matrix as candidates for  $\theta$  and select the value that maximizes some *score* on the the validation set. This *score* is generally defined based on the relative business value of the newly added intent class. For example; if the gain for correctly predicting examples in the newly added intent class is  $\alpha$  times higher than the gain of correctly predicting any other class then a simple formulation for the score can be

$\text{score} = \alpha * \text{number of examples correctly classified to } c_k + \text{number of examples correctly classified to any other class}$  We defined a more intuitive *normalized score* as the ratio of this score over the total possible score in the test set as follows:

$$\begin{aligned} \text{normalized score} &= \frac{\text{score}}{\text{total possible score}} \text{ where;} \\ \text{total possible score} &= \alpha * (\# \text{ examples in } c_k) \\ &+ (\# \text{ examples in any other class}) \end{aligned} \quad (4)$$

When  $\alpha = 1$ , *normalized score* equals *accuracy* and thus the grid search returns the value of  $\theta$  that maximizes the validation accuracy.

**4.2.3 Extending to multiple intents.** We also simulated an scenario when multiple new intents are added to the classifier. Instead of selecting a single intent, we randomly selected five intents  $\{c_1, c_2, c_3, c_4, c_5\}$  using the same criteria described in the previous section. The training procedure remains the same, but we slightly modified the inference algorithm to incorporate multiple "one vs. rest" classifiers. Details are described in algorithm 2 below.

---

### Algorithm 2 Inference Multiple:

---

Calculate the best combination of dist thresholds  $\{\theta_1, \theta_2, \dots, \theta_5\}$  for classes  $\{c_1, c_2, \dots, c_5\}$   
 Transform input  $x$  to embedding vector,  $v = f\phi(x)$   
**for**  $c_k \leftarrow \{c_1, c_2, c_3, c_4, c_5\}$  **do**  
   Calculate distance between  $v$  and all training examples belonging to intent class  $c_k$ ;  $d_k = \text{dist}(v, v_{train}), v_{train}$  in class  $c_k$  followed by average distance  $d_k = \text{average}(d_k)$   
   If  $d_k < \theta_k$ , add  $c_k$  to candidate classes  
**end for**  
**if** There are multiple candidate classes **then**  
   Assign example to  $c_k$  from the candidate classes with minimum ratio of  $d_k/\theta_k$   
**else**  
   Pass the example through baseline classifier to get the class label  
**end if**

---

The threshold  $\theta$  selection follows a similar procedure to the one described in Algorithm 1 except that the grid search evaluates the score for all possible combinations of  $\theta$ . Each selected intent can have a different  $\theta$ . So, the objective is to find the combination  $(\theta_1, \theta_2, \dots, \theta_5)$  that gives the highest score in the validation set. The complexity of this optimization grows exponentially with number of selected intents. Using the same candidate selection strategy as described for 1 there is a total of  $10^5$  combinations to choose from. A flowchart illustrating the Algorithm 2 and the threshold selection strategy explained above is shown in figure 3.

## 5 EMPIRICAL RESULTS AND ANALYSIS

For the rest of the paper **Baseline** refers to the baseline model that is a shallow neural network with two fully connected layers after taking the output from the universal sentence encoder (USE) as inputs, followed by a softmax activation function to produce a probability distribution over the all the intent classes described in subsection 4.1. **B+T** denotes the ensemble of the baseline classifier and the triplet network and **B+P** denotes the ensemble of the baseline and the Prototypical network using the inferences strategies described in Algorithms 1 and 2.

Our goal for the ensemble is to boost performance in the newly added intent class. We want to correctly classify more examples in the newly added intent class (i.e. improve recall of that class) but at the same time, it is important that the ensemble does not negatively affect the overall test accuracy. That's why we report recall in the newly added class as well as overall test accuracy. Table 1 shows recall of different classifiers on the selected new intent class on ACID dataset. The table shows that for the majority of the runs, the **Baseline** multiclass classifier has below average performance on

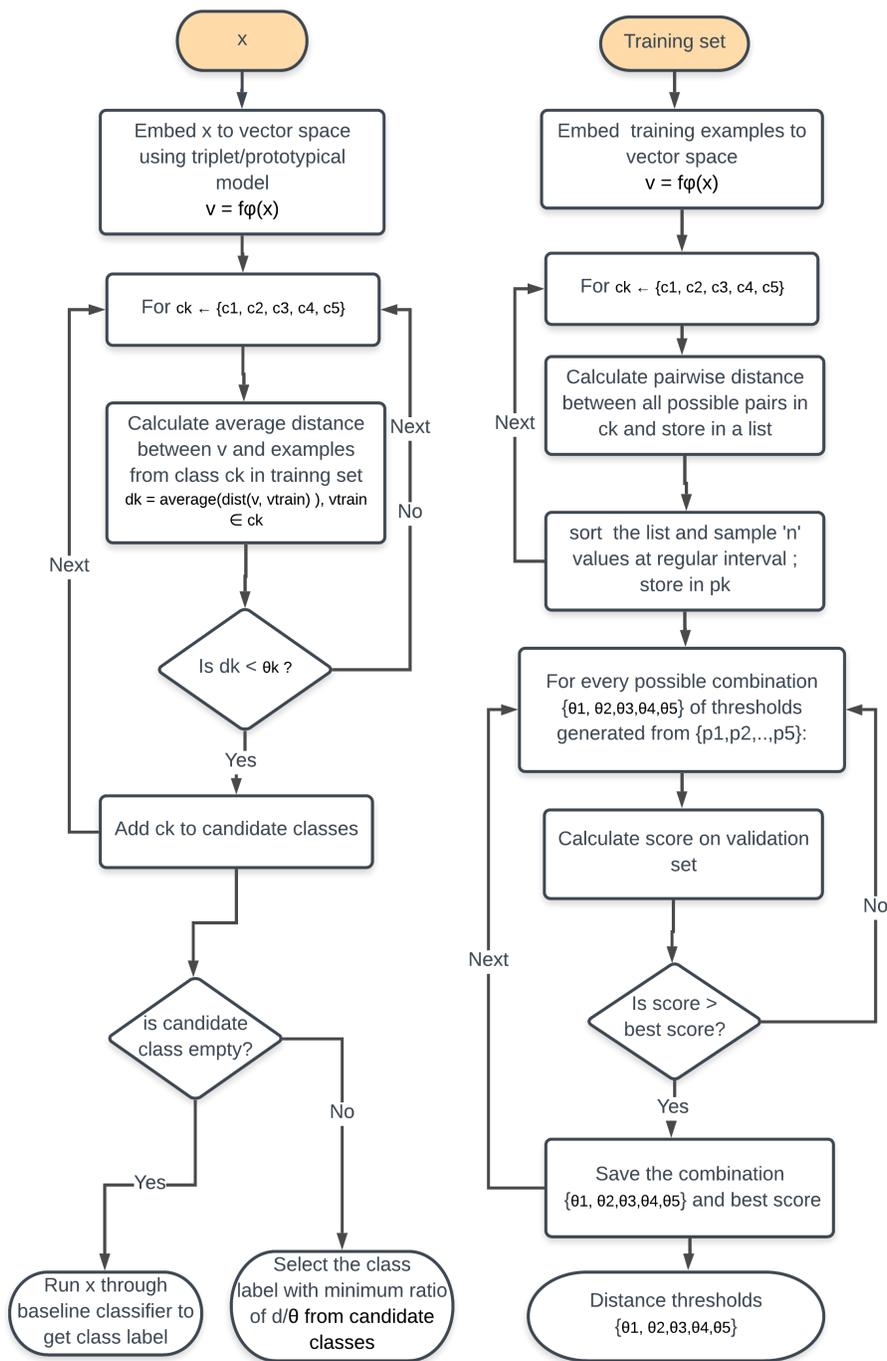


Figure 3: Left: Flowchart of Algorithm 2 and; Right: Flow chart of threshold selection strategy for Algorithm 2

the selected new intent class compared to the overall accuracy of the rest of the test set. It is evident that a traditional DNN multiclass classifier performs better on intents (classes) with larger number of examples and tends to perform poorly on the new intent classes with few examples. For example, for run 6 the **Baseline** model

correctly predicts 23% of the examples from selected intent class but the overall test accuracy for that run is much higher at 87.7%.

From Table 1 and 2, we also see that the ensemble gives a boost in performance for the selected intent class. We see this improvement on all the 10 runs. The improvement is very significant in

Run	INTENT	Intent Description / Reason	Recall on selected intent			Test Accuracy		
			Baseline	B+T	B+P	Baseline	B+T	B+P
1	INFO_PREPAID_CARD_PYMT	Info about credit card payment	0.409	0.909	<b>0.945</b>	0.881	<b>0.885</b>	0.883
2	INFO_AGT_NOT_RESPONDING	Agent is not responding to customer	0.837	0.893	<b>0.937</b>	0.885	<b>0.886</b>	<b>0.886</b>
3	INFO_SET_UP_ACCT	Info to set up an online Account	0.536	0.788	<b>0.855</b>	<b>0.881</b>	<b>0.881</b>	0.88
4	INFO_AMT_DUE	Info about Amount Due	0.358	0.45	<b>0.647</b>	0.877	0.879	<b>0.881</b>
5	INFO_RIDESHARE_COV	Info about coverage during rideshare	0.873	0.879	<b>0.93</b>	<b>0.886</b>	<b>0.886</b>	<b>0.886</b>
6	INFO_CL_CLAIM_FILED	Info about a filed claim	0.231	<b>0.705</b>	0.692	0.877	0.878	<b>0.88</b>
7	INFO_TRANSFER_ACCT_BAL	Info on account balance transfer	0.771	0.814	<b>0.888</b>	0.885	<b>0.886</b>	0.885
8	INFO_FORGOT_USERID	Info to retrieve forgotten user id	0.485	<b>0.86</b>	0.807	0.879	<b>0.883</b>	<b>0.883</b>
9	INFO_DIFFERENT_AMTS	Info about change in billing amount	0.578	0.584	<b>0.705</b>	0.878	0.878	<b>0.879</b>
10	INFO_ERS	Info on Emergency Roadside Service	0.358	0.526	<b>0.665</b>	0.871	<b>0.876</b>	0.875

Table 1: Result of 10 random experiments on ACID dataset each time selecting a different random intent. B+T represents ensemble of Baseline classifier and Triplet; B+P represents ensemble of baseline and Prototypical.

Run	INTENT	Recall on chosen intent			Test Accuracy		
		Baseline	B+T	B+P	Baseline	B+T	B+P
1	apr	0.525	<b>0.861</b>	0.686	0.936	<b>0.941</b>	0.938
2	ingredients_list	0.481	<b>0.861</b>	0.832	0.937	<b>0.941</b>	0.939
4	calendar	0.525	<b>0.729</b>	0.671	0.935	<b>0.938</b>	0.934
5	smart_home	0.678	<b>0.861</b>	0.708	0.94	<b>0.942</b>	0.94
6	freeze_account	0.649	0.788	<b>0.817</b>	0.937	<b>0.939</b>	<b>0.939</b>
7	alarm	0.897	<b>0.97</b>	0.927	0.942	<b>0.943</b>	0.942
8	make_call	0.437	0.7	<b>0.729</b>	0.935	0.937	<b>0.94</b>
9	tell_joke	0.824	<b>0.963</b>	0.824	0.941	<b>0.944</b>	0.941
10	order_checks	0.693	<b>0.941</b>	0.751	0.939	<b>0.943</b>	0.939

Table 2: Result of 10 random experiments on CLINC150 dataset each time selecting a different random intent. B+T represents ensemble of Baseline classifier and Triplet; B+P represents ensemble of baseline and Prototypical.

Run	Model	c1	c2	c3	c4	c5	Test Acc.
1	Base	0.583	0.225	0.287	0.0754	0.417	0.83
	B+T	<b>0.644</b>	<b>0.725</b>	<b>0.594</b>	0.287	<b>0.552</b>	<b>0.846</b>
	B+P	0.605	0.621	0.504	<b>0.349</b>	0.441	0.842
2	Base	0.421	0.703	0.088	0.672	0.602	0.851
	B+T	0.692	0.726	<b>0.417</b>	<b>0.867</b>	0.725	<b>0.863</b>
	B+P	<b>0.739</b>	<b>0.73</b>	0.341	0.723	<b>0.774</b>	0.858
3	Base	0.681	0.282	0.607	0.639	0.75	0.858
	B+T	<b>0.782</b>	<b>0.517</b>	<b>0.818</b>	<b>0.737</b>	<b>0.898</b>	<b>0.87</b>
	B+P	0.743	0.464	0.747	0.724	0.804	0.866
4	Base	0.564	0.491	0.168	0.821	0.238	0.846
	B+T	<b>0.688</b>	<b>0.664</b>	<b>0.471</b>	<b>0.839</b>	<b>0.446</b>	<b>0.855</b>
	B+P	0.62	0.572	0.425	0.833	0.291	0.852
5	Base	0.531	0.577	0.225	0.164	0.436	0.844
	B+T	<b>0.593</b>	<b>0.627</b>	0.506	<b>0.294</b>	0.547	<b>0.855</b>
	B+P	0.564	0.598	<b>0.554</b>	0.235	<b>0.563</b>	0.853

Table 3: Result of 5 random experiments on ACID dataset each time selecting a different (random) set of 5 intent classes(c1,c2,c3,c4,c5). Columns c1 through c5 present recall score on selected intent classes and the last column presents overall Test accuracy. B+T represents the ensemble of the Baseline classifier and the triplet network; B+P represents ensemble of Baseline and the prototypical network.

some of the runs. However, it is important to ensure that higher recall in selected intent class does not lead to reduced overall test

accuracy due to false positives in that class. So, the desired outcome is increase in recall on that class without affecting the overall test

Run	Model	c1	c2	c3	c4	c5	Test Acc
1	Baseline	0.234	0.796	0.613	0.73	0.781	0.918
	B+T	0.774	<b>0.891</b>	<b>0.898</b>	<b>0.964</b>	<b>0.898</b>	<b>0.933</b>
	B+P	<b>0.781</b>	0.861	0.81	0.869	0.861	0.924
2	Baseline	0.588	0.781	0.912	0.219	0.723	0.92
	B+T	<b>0.816</b>	<b>0.898</b>	<b>0.956</b>	<b>0.861</b>	<b>0.964</b>	<b>0.933</b>
	B+P	0.809	0.818	0.934	0.832	0.92	0.926
3	Baseline	0.788	<b>0.985</b>	0.62	0.825	0.92	0.932
	B+T	<b>0.964</b>	<b>0.985</b>	<b>0.934</b>	<b>0.942</b>	<b>0.934</b>	<b>0.943</b>
	B+P	0.912	<b>0.985</b>	0.818	0.898	0.92	0.935
4	Baseline	0.672	0.131	0.883	0.569	0.788	0.917
	B+T	<b>0.839</b>	<b>0.715</b>	0.971	0.803	<b>0.905</b>	<b>0.932</b>
	B+P	0.781	0.693	<b>0.978</b>	<b>0.818</b>	0.825	0.928
5	Baseline	0.832	0.504	0.693	0.898	0.146	0.918
	B+T	<b>0.978</b>	0.832	<b>0.934</b>	<b>0.956</b>	<b>0.839</b>	<b>0.935</b>
	B+P	0.876	<b>0.847</b>	0.861	0.942	0.781	0.924

Table 4: Result of 5 random experiments on CLINC150 dataset each time selecting a different (random) set of 5 intent classes(c1,c2,c3,c4,c5). Columns c1 through c5 present recall score of selected intent classes and the last column presents overall Test accuracy. B+T represents the ensemble of the Baseline classifier and the triplet network; B+P represents ensemble of Baseline and the prototypical network.

Run	Model	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
1	Baseline	0.83	0.791	0.758
	B+T	0.846	0.824	0.805
	B+P	0.842	0.817	0.795
2	Baseline	0.851	0.819	0.792
	B+T	0.863	0.845	0.831
	B+P	0.858	0.84	0.824
3	Baseline	0.858	0.836	0.818
	B+T	0.87	0.861	0.853
	B+P	0.866	0.853	0.841
4	Baseline	0.846	0.813	0.786
	B+T	0.855	0.836	0.819
	B+P	0.852	0.831	0.813
5	Baseline	0.844	0.803	0.772
	B+T	0.855	0.834	0.815
	B+P	0.853	0.814	0.791

Table 5: Normalized score for different values of  $\alpha$  for the five different runs corresponding to Table 3 ACID dataset. Higher value of  $\alpha$  represents that the newly added intent is deemed more important. B+T represents the ensemble of the Baseline classifier and the triplet network; B+P represents ensemble of Baseline and the prototypical network.

accuracy. For example; for run 1 in table 1 the recall of selected intent class improved from 0.409 with baseline to 0.909 with B+T and the overall test accuracy also improved from 88.1% to 88.5%. We can see that the overall test accuracy either improves marginally or remains unchanged in all the runs while the improvement in selected intent class is significant which is often the desired outcome for the newly added intents.

Table 3 and 4 present results for five random experiments representing a scenario where five new intents  $\{c_1, c_2, \dots, c_5\}$  are added to the CA. We observe improvement in both the recall score and the overall test accuracy for all five new intent classes with our proposed ensemble classifier. These improvements are consistent for all five runs. The rationale behind adding 5 new intents was based on some preliminary experiments which showed that beyond 5 intents, it was hard to gain improvement in recall without affecting the test accuracy.

The results presented in Table 1-4 are special cases of our proposed approach where the distance threshold  $\theta$  of the ensemble is tuned to optimize test accuracy. As described in the previous section, the distance threshold  $\theta$  can be optimized to maximize a normalized score (equation 4) in order to give more importance to newly added intents  $\{c_1, \dots, c_5\}$ .

The importance associated with correctly classifying examples to one of these five classes can be represented by the weighting factor  $\alpha$  which is often determined according to business needs. A higher value of  $\alpha (> 1)$  represents that the new intent class is more important than the existing intents.

To illustrate this, Table 5 reports *normalized scores* for three different values of  $\alpha$  for the five runs corresponding to Table 3 (ACID dataset). At  $\alpha = 1$ , the *normalized scores* is equivalent to accuracy and equals to result from table 3. As the value of  $\alpha$  increases, the boost in performance resulting from the ensemble is more significant. For instance; at  $\alpha = 3$  we see improvement of 3% to 5% on the normalized score across all five runs.

Table 2 and 4 shows results for the CLINC150 dataset for the same set of experiment as ACID dataset in tables 1 and 3 respectively. Both table 2 and 4 show that recall on newly added intent class(es) increases for all the runs and the overall test accuracy either improves or remains unchanged. This result is consistent

with the results from ACID dataset, thereby showing that the approach can be applied to any intent classification domain to solve the problem of adding "new intent" and/or "prioritizing new intent" categories with few examples.

## 6 CONCLUSIONS AND FUTURE WORK

We show that a simple but effective combination of an embedding transformation model and a standard neural network multiclass classifier can achieve significant improvements in performance on newly added intents for which training examples are scarce while maintaining and sometimes improving the overall intent classifier performance. This ensemble approach can be easily optimized to maximize performance on a subset of intent classes that are deemed important by business needs. Results in a publicly available dataset that contains intents from different industries demonstrate that the proposed approach can be easily extended to other domains.

Our approach is easy to implement and could be easily combined or appended to current deployed CA implementations making it very practical in most industrial settings.

As future work, we would like to explore creating a system that can automatically decide when enough samples are provided to transfer intents to be part of the main neural network classifier in order to maximize accuracy. This would be a key piece towards a self-maintained intent classification system. Another topic of interest for future work is how recent existing state-of-the-art language representations (like BERT [3]) perform with combines with our proposed approach. We also would like to share insights about post-deployment evaluation after updating the deployed system with several cycles of new added intent classes.

## REFERENCES

- [1] Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. 2020. Efficient Intent Detection with Dual Sentence Encoders. *arXiv preprint arXiv:2003.04807* (2020).
- [2] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.
- [4] Jason Krone, Yi Zhang, and Mona Diab. 2020. Learning to Classify Intents and Slot Labels Given a Handful of Examples. *arXiv:cs.CL/2004.10793*
- [5] Varun Kumar, Hadrien Glaude, Cyprien Lichy, and William Campbell. 2019. A Closer Look At Feature Space Data Augmentation For Few-Shot Intent Classification. 1–10. <https://doi.org/10.18653/v1/D19-6101>
- [6] 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. 2019. 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)*. <https://www.aclweb.org/anthology/D19-1131>
- [7] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.
- [8] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*. 4077–4087.
- [9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.
- [10] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in neural information processing systems*. 3630–3638.
- [11] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. 2006. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*. 1473–1480.

## A APPENDIX

In this section we provide details of network parameters used during training.

### A.1 Baseline Model Parameters

As explained in paper, we create Baseline Model by adding two fully connected(FC) layers to output of transformer based Universal Sentence Encoder(USE). We use the transformer-based USE en-coder [9] which targets high accuracy at the cost of greater model complexity and resource utilization [2]. We keep the USE weights fixed during training. We use the pretrained model available in tensorflowhub. Hidden layer FC1: 512 units, activation=relu, dropout ratio = 0.25 Output layer FC2: 250 units, activation = softmax Learning rate = 5e-5 Batch size = 32 Number of epochs: 50 Loss = categorical crossentropy Hyper parameter tuning: We did hyperparameter search on number of hidden layers, hidden layer dimension and learning rate.

### A.2 Triplet Network Parameters

As explained in the paper, we create the triplet network by adding two fully connected (FC) layers to Universal Sentence Encoder(USE) output. Hidden layer FC1: 512 units, activation= relu, dropout ratio = 0.25 Output layer FC2: 250 units, activation = None, normalization= l2 normalization Learning rate = 5e-5 Batch size = 1000 (Large batch size suggested in the paper [7]) Number of steps: 5000 Loss = triplet loss Triplet Selection Strategy: Random triplet mining We did hyperparameter search on number of hidden layers, hidden layer dimension, learning rate and triplet selection strategy. We tried Random Triplet mining, Hard triplet mining and semi-hard triplet mining as suggested in the paper but did not observe difference in performance. Rest of the Training procedure is identical to the one described in the Facenet paper [7]

### A.3 Prototypical Network Parameters

As explained in the paper, we create the prototypical network by adding two fully connected (FC) layers to Universal Sentence Encoder(USE) output. Hidden layer FC1: 250 units, activation= relu, dropout ratio = 0.25 Output layer FC2: 250 units, activation = None, normalization= l2 normalization Learning rate = 5e-4 Number of steps: 5000 N\_way = 20 N\_query = 8 N\_shot=5 We did hyperparameter search on number of hidden layers, hidden layer dimension, learning rate, N\_way, N\_query, N\_shot. Please refer to paper [8] for more explanation on these parameters. Rest of the Training procedure is identical to the one described in the paper.