ICON 2020

17th International Conference on Natural Language Processing

Proceedings of the Workshop

December 18 - 21, 2020
Indian Institute of Technology Patna, India

Introduction

Welcome to the ICON 2020 Workshop on Joint NLP Modelling for Conversational AI.

Virtual Assistants/Dialog Systems are spanning from cloud-based systems to embedded devices formulating a hybrid architecture for several reasons such as privacy of the data, faster processing, offline interaction, personalization etc. There are several NLP tasks running as part of the Virtual Assistant example Domain Classification, Intent Determination, Slot Extraction, Dialog Management, Natural Language Generation, Text to Speech and Automatic Speech Recognition. Thus, in both cloud and embedded devices, it is necessary to combine various NLP tasks to improve latency and memory usage. Additionally, a joint NLP model is suitable for both cloud and embedded devices processing.

Virtual Assistants/Dialog Systems are also capable of learning by themselves from various information accessible to it. For example, contextual data, personal data, queries, unhandled or failed queries etc. It is a very interesting area to explore how a joint model can incrementally learn from the data available to the Virtual Assistant.

The goal of this workshop is to bring together NLP researchers and practitioners in different fields, alongside experts in speech and machine learning, to discuss the current state-of-the-art and new approaches, to share insights and challenges, to bridge the gap between academic research and real-world product deployment, and to shed the light on future directions. "Joint NLP modelling for Conversational AI" will be a one-day workshop including keynotes, spotlight talks, posters, and panel sessions. In keynote talks, senior technical leaders from industry and academia will share insights on the latest developments of the field. An open call for papers will be announced to encourage researchers and students to share their prospects and latest discoveries. The panel discussion will focus on the challenges, future directions of conversational AI research, bridging the gap in research and industrial practice, as well as audience suggested topics.

We received 16 submissions, and after a rigorous review process, we only accepted 6 papers. The overall acceptance rate for the workshop was 37.5%.

Joint NLP modelling for Conversational AI ICON 2020 Organizers
Ranjan Kumar Samal, Bixby AI, Samsung Electronics
Praveen Kumar GS, Bixby AI, Samsung Electronics
Siddhartha Mukherjee, Bixby AI, Samsung Electronics

Organizers

Ranjan Kumar Samal - Architect of NLU, Samsung Research Institute-Bangalore
Praveen Kumar GS - Director of NLU, Samsung Research Institute-Bangalore
Siddhartha Mukherjee - Engg. Leader of NLU, Samsung Research Institute-Bangalore

Program Committee:

- Members from Samsung R&D Institute Bangalore
  - Gaurav Mathur
  - Anish Nediyachanth
  - Bharatram Natarajan
  - Dr. Priyadarshini Pai
  - Dr. Sandip Bapat
  - Dr. Jayesh Mundayadan Koroth
  - Dr. Nitya Tiwari
  - Dr. Hardik Bhupendra Sailor
  - Dr. Shekhar Nayak
  - Dr. Mahaboob Ali Basha Shaik
- Members from Samsung Electronics, Suwon, South Korea
  - Sojeong Ha
  - Dr. Jun-Seong Kim
- Members from IIT-Patna
  - Dr. Asif Ekbal, Professor, Department of Comp Sc & Engg, IIT Patna
  - Dr. Sriparna Saha, Professor, Department of Comp Sc & Engg, IIT Patna
- Members from IIT-Palakkad
  - Dr. Mrinal kanti Das, Assistant Professor, CSE, IIT Palakkad
- Members from POSTECH, South Korea
  - Dr. Jinwoo Park, CSE, POSTECH, Republic of Korea

Table of Contents

Workshop Program

Monday, December 21, 2020

10:00 ***Opening Note*** *by Praveen Kumar G S, Director and Head of NLU, SRIB*

10:15 ***"Transformation from NLU to Multimodal NLU"*** *by Siddhartha Mukherjee, Staff Engineering Manager of NLU, SRIB*

11:00 ***Neighbor Contextual Information Learners for Joint Intent and Slot Prediction***
Bharatram Natarajan, Gaurav Mathur and Sameer Jain

11:30 ***Unified Multi Intent Order and Slot Prediction using Selective Learning Propagation***
Bharatram Natarajan, Priyank Chhipa, Kritika Yadav and Divya Verma Gogoi

12:00 ***EmpLite: A Lightweight Sequence Labeling Model for Emphasis Selection of Short Texts***
Vibhav Agarwal, Sourav Ghosh, Kranti CH, Bharath Challa, Sonal Kumari, Harshavardhana and BARATH RAJ KANDUR RAJA

12:30 ***Optimized Web-Crawling of Conversational Data from Social Media and Context-Based Filtering***
Annapurna P Patil, Rajarajeswari Subramanian, Gaurav Karkal, Keerthana Purushotham, Jugal Wadhwa, K Dhanush Reddy and Meer Sawood

13:00 ***Lunch Break***

14:00 ***AI for real-world and AI for future; Panel Discussion by Senior Thought Leaders at SRIB*** *by Praveen Kumar GS, Ratnakar Rao VR, Sreevatsa DB, Ritesh Jain, Srinivasa Rao Kudavelly*

15:00 ***Named Entity Popularity Determination using Ensemble Learning***
Vikram Karthikeyan, B Shrikara Varna, Amogha Hegde, Govind Satwani, Shambhavi B R, Jayarekha P and Ranjan Samal

15:30 ***"Discovering Subtle Information from Texts using Bayesian Models"*** *by Dr. Mrinal Kanti Das, Assistant Professor, CSE, IIT Palakkad*

16:30 ***A character representation enhanced on-device Intent Classification***
Sudeep Deepak Shivnikar, Himanshu Arora and Harichandana B S S

17:00 ***Closing Note*** *by Ranjan Samal, Architect of NLU, SRIB*

Neighbor Contextual Information Learners for Joint Intent and Slot Prediction

Bharatram Natarajan[*], **Gaurav Mathur**[*] and **Sameer Jain**
research.samsung.com
{bharatram.n, gaurav.m4, sameer.jain}@samsung.com

Abstract

Intent Identification and Slot Identification are two important task for Natural Language Understanding (NLU). Exploration in this area have gained significance using networks like RNN, LSTM and GRU. However, models containing the above modules are sequential in nature, which consumes lot of resources like memory to train the model in cloud itself. With the advent of many voice assistants delivering offline solutions for many applications, there is a need for finding replacement for such sequential networks. Exploration in self-attention, CNN modules has gained pace in the recent times. Here we explore CNN based models like Trellis and modified the architecture to make it bi-directional with fusion techniques. In addition, we propose CNN with Self Attention network called Neighbor Contextual Information Projector using Multi Head Attention (NCIPMA) architecture. These architectures beat state of the art in open source datasets like ATIS, SNIPS.

1 Introduction

Intelligent Voice Assistant like Samsung Bixby, Google Assistant, Amazon Alexa and Microsoft Cortana are increasingly becoming popular. These assistants cater to the user request by extracting the user intention from the spoken utterance. NLU express the user intention in terms of intent label and slot tags. There is one intent label for entire utterance, which signifies unique action to execute. Whereas slots are tags given to tokens in utterance, which signifies extra information required to execute the unique action. Slot tags are denoted in IBO format. For example, consider the utterance "what flights are available from Denver to San Francisco". We denote Intent label as "atis_flight". We denote Slot information as "O O O O O B-fromloc.city_name O B-toloc.city_name I-toloc.city_name". There is one slot tag for every token in the utterance, where "O" represents that token is not a slot and B-fromloc.city_name represents that token is "Beginning of slot fromloc.city_name" and "B-toloc.city_name I-toloc.city_name" represents that corresponding tokens are Beginning and Continuation of slot toloc.city_name respectively. We consider Intent Identification as Classification task and Slot tagging as sequence labelling task where we predict slot for each word.

Lots of work has happened in this area. Initially, research community explored both intent and slot tasks as independent task. Different techniques were explored for intent classification. Firdaus et al. (2018) created ensemble model by combining learnings of CNN, LSTM and GRU using multilayer perceptron to enhance intent classification. Kim et al. (2016) proposed enriched word embedding for making similar words together and dissimilar words farther, which aided intent detection better. Yolchuyeva et al. (2020) proposed use of self-attention for enhancing intent classification by capturing long-range and multi-scale dependency in data.

Similarly for slot tagging, Kurata et al. (2016) proposed use of label dependencies along with input sentence for enhancing slot learnings using LSTM. Mesnil et al. (2014) suggested use of Recurrent Neural Network for slot tagging task. Ngoc Thang Vu (2016) proposed novel CNN architecture for slot tagging task, which also used past and future words information.

These individual models approach resulted in pipelined design of NLU. Where intent model will predict intent and they use intent output in slot model to predict slot. This resulted in error propagation i.e. error in intent output resulted in slot tagging errors. In addition, the intent and slot learnings are not available for each other to enhance

[*]Authors contributed equally

Proceedings of the ICON 2020 Workshop on Joint NLP Modelling for Conversational AI, pages 1–9
Patna, India, December 18 - 21, 2020.

each task learning.

To circumvent the above limitation, research community started exploring unified model where they identify both intent classification and slot tagging. Liu and Lane (2016) proposed attention as RNN encoder as input to Decoder for predicting slot and intent by the decoder. Tingting et al. (2019) proposed exploration of attention with Bi-LSTM for joint intent and slot prediction. Firdaus et al. (2019) suggested usage of CNN and RNN for contextual understanding of the utterance along with CRF for label dependency to predict intent and slot. Hardalov et al. (2020) proposed intent pooling attention along with word features on top of BERT for predicting intent and slot. The above approaches addresses both the task using single unified network where both the learnings are propagated to single network.

Further, exploration of parallel learning networks, using common base module, with fusing of intent and slot learnings have gained momentum. Goo et al. (2018) suggested slot gate mechanism to fuse the intent attention learning with slot attention learning to predict slot along with intent. E et al. (2019) proposed a novel iteration mechanism to fuse the meanings of intent and slot subnet for predicting intent and slot.

Inspired by the work on parallel learning networks, where we address the intent and slot tasks learnings by each parallel network, we are proposing a novel architecture, Neighbor Contextual Importance Projector (NCIP) for learning the word importance in its immediate vicinity to boost its importance learning in the overall utterance. We use parallel Multi head attention on top of NCIP to project importance phrases for each task, to predict intent and slot. We are also exploring Trellis network, which learns immediate neighbors in a layer and entire utterance in multiple such layers. In addition, the weights of the network is shared in both temporal direction as well as across layers. Hence, we are proposing a bi-directional Trellis network with different fusion techniques, like Linear Fusion, Concatenation, to predict intent and slot to mimic bi-directional LSTM or GRU.

We organize rest of the section as follows. Section 2 describes the Proposed Approaches. Section 3 describes the experimental setup including dataset, metrics used followed by results. Finally, we conclude and suggest future work and extensions.

2 Proposed Approaches

2.1 Trellis Network Based Architectures

Bai et al. (2018) proposed a novel architecture containing special Temporal Convolutional Neural Networks (T-CNN) for language modeling (LM) task. In this model, they share weights across all layers and they inject the input to deep layers.

As Trellis network has structural and algorithmic elements from both LSTM & CNN, it achieved state of art in various LM tasks. Therefore, we are exploring extension of the same in intent determination and slot tagging.

Original Trellis Network process the input sequence in forward direction only. We propose bidirectional network with two parallel Trellis network encoders, one for forward pass and one for backward pass, for intent determination and slot tagging.

2.1.1 Utterance Pre-Processing

We tokenize the input utterance into words. If the number of words is less than seq_len (seq_len is obtained by taking maximum of the count of words for each utterance in the dataset), we pad that utterance with "PAD" word. We convert words to index using dictionary of unique training words to incremental index. If we do not find any word in the dictionary, than assign the index of "unk" (specially added token into dictionary to handle unseen words). We convert every utterance of training batch into list of indices to generate training data of shape [bs, seq_len]. Where 'bs' is batch size.

For bidirectional Trellis network, we first reverse the training utterances than apply same preprocessing as on original utterances.

2.1.2 Unidirectional Trellis Network Based Architecture

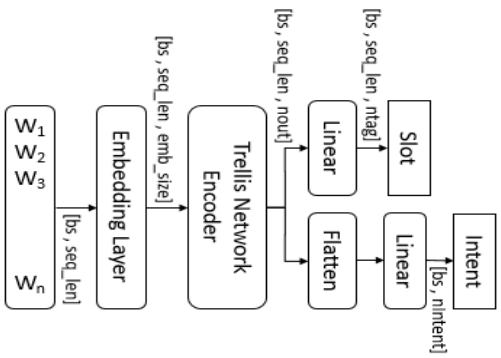

Figure 1: Unidirectional Trellis flow for joint intent and slot prediction

Figure 1 shows unidirectional Trellis encoder based architecture. We pass input data of shape [bs, seq_len] through embedding layer to represent every word with embedding size (emb_size) vector. Trellis Network is used as an encoder to represent input data in [bs, seq_len, nout] dimension. Where nout is a hyper parameter of Trellis network. For slot prediction, output of Trellis network is passed by a linear layer to produce output of shape [bs, seq_len, ntag], where ntag is number of tags.

For intent determination, we first flatten the Trellis output than pass to output linear layer. It produces output of shape [bs, nIntent], where nIntent is number of intent labels.

2.1.3 Bidirectional Trellis Network Based Architecture

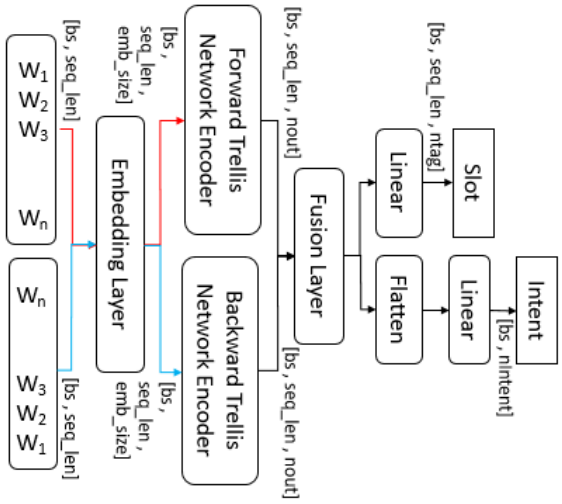

Figure 2: Bidirectional Trellis Flow for joint intent and slot prediction

Figure 2 shows Bi-directional Trellis network. In this, we used two Trellis network encoders, first to process the input sequence as-is and the second to process a reversed copy of the input sequence. Both Trellis encoders generates outputs of dimension [bs , seq_len , nout] . Then we pass both the Trellis network outputs through fusion layer. Fusion layer determines the importance of each Trellis output by selectively propagating the learnings from both the outputs. Then we predict intent by flattening the fused output and passing through dense layer with nintent [distinct intents] as final labels. We also predict slot by passing through dense layer with ntag [distinct tags] as output label. We explore two fusion techniques in Section 2.1.4 and 2.1.5 and Trellis network in 2.1.6.

2.1.4 Masked Flip and Concatenation

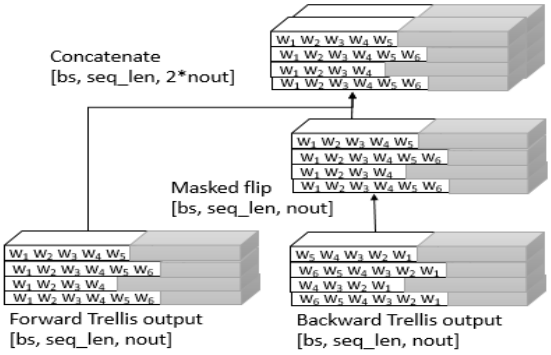

Figure 3: Masked flip and concatenation

Masked flip is implemented by Gardner et al. (2017) in AllenNLP platform. As shown in Figure 3, we first flip the output of backward Trellis (it brings representation of "pad" at beginning) than slice every example of batch on original unpadded length of that example. It gives representation of pad and actual words separately, than again concatenate pad representation at end of word representation. After masked flipping of backward Trellis output, concatenate it with forward Trellis output.

2.1.5 Linear Fusion

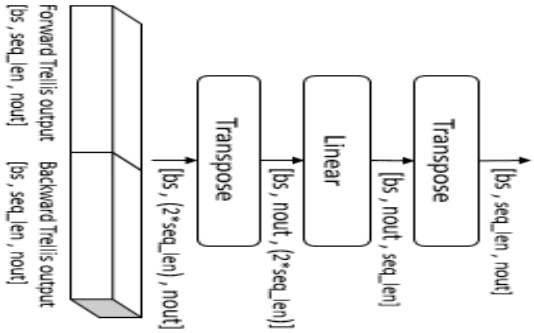

Figure 4: Linear Fusion

As shown in Figure 4, we are using a linear layer to learn joint representation of forward and backward Trellis outputs. For it we first concatenate both outputs at axis 1, then transpose it to bring word representation as final axis and pass by linear layer to bring the final axis to seq_len. We then transpose the same to get seq_len in first axis.

2.1.6 Trellis Network Decoded

As we are using Trellis Network to learn a token representation from its neighbor, let us discuss

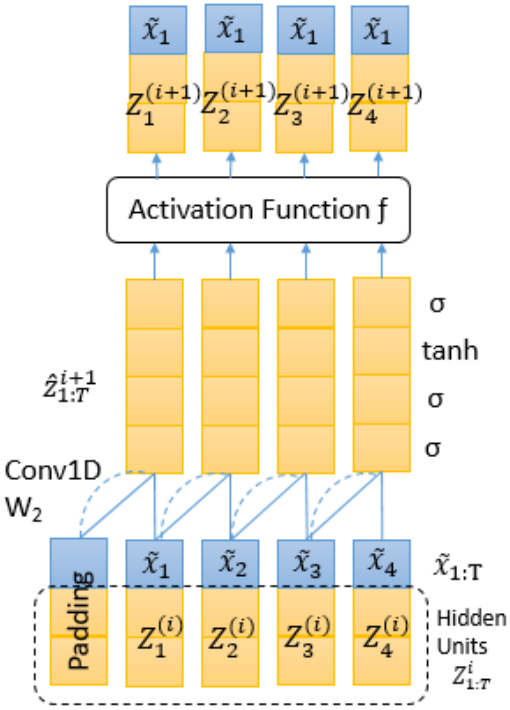

Figure 5: Inter layer transformation of Trellis

about its working. Trellis network is a special form of temporal convolutional neural network (T-CNN) with special structure. It share weights across depth and provide input matrix to all the layers. Figure 5 explains the working of Trellis in intermediate layers. We denote $x_t \epsilon \mathbb{R}_p$ as input embedding vectors at time step t. We denote $Z_1^{T_i} \epsilon \mathbb{R}_q$ as hidden output vector for all-time steps 1 to T and layer i. Hyper parameters of Trellis networks are nhid is hidden size , nout is output size and hsize = nhid + nout

Hidden output $Z_{1:T}^{i+1}$ at for layer i+1 is computed by three steps

First, precompute linear transformation on input x, result will be directly passed to all layers as shown in Equation 1.

$$\tilde{x}_{1:T} = Conv1D(x_{1:T}, W_1) \tag{1}$$

Shape of input x is (bs x emb_size x seq_len) which is transpose of embedding layer output. Shape of W1 is [4* hsize, emb_size, kernel size). Kernel size is fixed to 2. nhid is hidden layer size. Padding is done to keep output as same seq_len as input, $\tilde{x}_{1:T}$ will be of shape [bs, (4* hsize), seq_len]

After computing Pre-activation output $\hat{z}_{1:T}^{i+1}$, it will be divided into four equal parts to apply LSTM style activation function.$\hat{z}_{1:T}^{i+1}$ is kind of considered as concatenation of input gate, output gate, cell state and forget gates. This is the reason number of filters are kept 4*hsize in convolution operations. Conv1D computes Pre-activation output $\hat{z}_{1:T}^{i+1}$ as shown in Equation 2.

$$\hat{z}_{1:T}^{i+1} = Conv1D(z_{1:T}^{i}, W_2) + \ddot{(}x)_{1:T} \tag{2}$$

Shape of previous layer hidden output $z_{1:T}^{i}$ is [bs , hsize , seq_len], which can be initialized all zero for first layer. Shape of W_2 is [4* hsize, hsize , kernel size]. Kernel size is fixed to 2. Padding is done to keep output as same seq_len as input. Pre-activation output $\hat{z}_{1:T}^{i+1}$ will be of shape [bs, 4*hsize , seq_len].

Lastly, we produce Output $z_{1:T}^{i+1}$ by non-linear activation function as shown in Equation 3.

$$z_{1:T}^{i+1} = f(\hat{z}_{1:T}^{i+1}, z_{1:T-1}^{i}) \tag{3}$$

As we discussed, the nonlinear activation is based on LSTM cell equations. The pre-activation output is equally divided in four parts of shape [bs , hsize , seq_len] as shown in Equation 4

$$\hat{z}_{1:T}^{i+1} = [\hat{z}_{1:T,1}^{i+1}, \hat{z}_{1:T,2}^{i+1}, \hat{z}_{1:T,3}^{i+1}, \hat{z}_{1:T,4}^{i+1}] \tag{4}$$

Output $z_{1:T}^{i+1}$ has two parts, computed as shown in Equation 5

$$\begin{aligned} z_{1:T,1}^{i+1} =& \sigma(\hat{z}_{1:T,1}^{i+1}) \odot z_{0:T-1,1}^{i} + \sigma(\hat{z}_{1:T,2}^{i+1}) \\ & \odot tanh(\hat{z}_{1:T,3}^{i+1}) \\ z_{1:T,1}^{i+1} =& \sigma(\hat{z}_{1:T,4}^{i+1}) \odot tanh(z_{1:T,1}^{i+1}) \end{aligned} \tag{5}$$

Therefore, in multiple such layers, Trellis network is learning short and long distance relation in input sequence. From last layer' $z_{1:T}^{i}$, last "nout" representations for every time stamp are passed as final output.

2.2 NCIPMA Network

Figure 6 shows Neighbor Contextual Information Projector with Multi Head Attention. (NCIPMA) architecture. First, we pass the utterance through Utterance Pre-Processing stage explained in 2.2.1.

2.2.1 Utterance Pre-Processing

The input utterance is broken down into chunk of words. If the number of words is less than required maximum number (obtained by taking maximum of the count of words for each utterance in the training data, denoted as max_words), we pad the rest of the words with "PAD" word. We convert words to index using sorted dictionary of unique

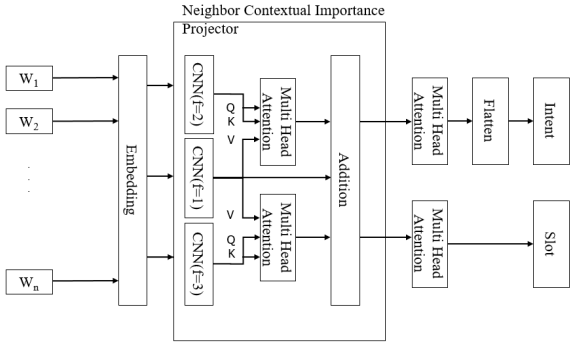

Figure 6: NCIPMA Architecture. Neighbor Contextual Importance Projector enhances the word importance learning by adding the unigram, bi-gram and tri-gram word importance learnings using Multi- Head Attention. Here f=1 means filter size as 1 representing unigram, f=2 means filter size as 2 representing bi-gram and f=3 means filter size as 3 representing tri-gram. MHA means Multi-Head Attention

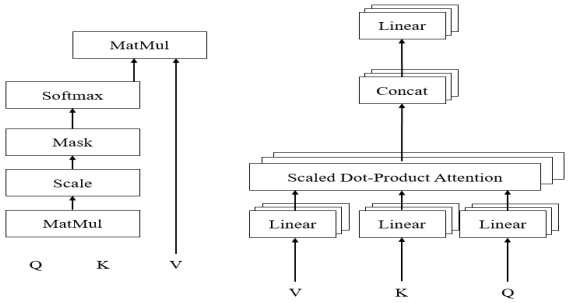

Figure 7: (Left) Scaled Dot-Product Attention. (Right) Multi-Head Attention consist of many scaled dot product attention in parallel.

training words to incremental index. If we do not find the word in the dictionary, then we assign the index of "unkword" (specially added into sorted dictionary to handle unseen words).We map sorted words in dictionary to index from 1 to n (number of words in the dictionary). We use index 0 for "PAD" word. This is the way we convert utterance to list of indices.

We pass the converted list of indices to Embedding layer. During training time, we have trained Embedding layer by masking zero index, passing weight-embedding matrix and making the matrix as trainable. We used unique training words, from sorted dictionary, to construct weight-embedding matrix. We did this by taking word index from sorting dictionary and 300-dimension word embedding from Glove Embedding for each word in unique training words. Hence, Embedding layer provides trained word embedding vector for each word index in the utterance during test time. This creates 3-d matrix of size (1, max_words, 300).

2.2.2 Neighbor Contextual Information Projector(NCIP) Module

In this module, we pass the 3-d matrix (output of utterance pre-processing module) through three parallel CNN layers. Each CNN layer uses filter size as one, two and three respectively capturing unigram, bi-gram and tri-gram word information. To learn each word information importance over other, we keep the word length same by making padding "same".

We capture the importance of uni-gram, bi-gram and tri-gram word information on the uni-gram word information using Multi-Head Attention, as shown in Figure 7. Multi-Head Attention aid in capturing multiple phrases importance in the provided input by passing the same input as query, key and value and calculating the importance as shown in Figure 6. Here, we pass n-gram (uni-gram, bi-gram or tri-gram) word information as query and key. We pass value as unigram word information.

Finally, we add the outputs of all the three Multi-Head Attention Module with unigram output.

This module aids in capturing

- Multi-phrase importance for each word.
- Multi-phrase importance for each phrase made from two to three words as well.

Addition of the above information projects that a word is important even as a part of the phrase and not only as a single word. We pass this information to two parallel Multi-Head Attention Modules.

These parallel Multi-Head Attention (MHA) modules try to learn the importance of phrases for each task in the provided input by passing the input as Query, Key and Value as shown in Figure 7. We predict intent through one MHA module by first flattening the 3 dimensional output, then by passing through dense layer with distinct intent as final hidden dimension. We predict slot through another MHA module, by passing through dense layer with distinct slot as final hidden dimension.

2.3 NCIPMA With CRF

Figure 8 shows the architecture of NCIPMA with CRF. First, we pass the utterance through Utterance Pre-Processing module. We pass the output of the module to NCIP module. We use the output of NCIP module to predict intent and slot.

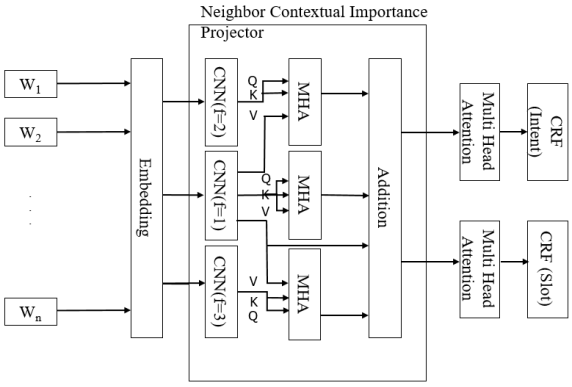

Figure 8: NCIPMA with CRF

For intent, we pass through MHA module to enhance the learning important for intent. Then we pass through Conditional Random Field (CRF) module.

We use linear chain CRF. Linear Chain CRF implements sequential dependencies during prediction. It is a generalization of Hidden Markov Model (HMM) where it solves the chain graph problem. CRF predicts for each word, the most probable intent possible. We then check if we predict the same intent for all the words and consider the intent as pass if we do so otherwise fail.

Similarly, for slot, we pass through MHA module to enhance the learning important for slot. Then we pass through Conditional Random Field (CRF) module. We use same linear chain CRF module. This module now predicts the most probable slot, for each word.

2.4 RASA DIET

RASA has developed a separate neural network framework and **??** proposed DIET architecture for intent classification and slot tagging. Impressed by the framework, we conducted experiments on RASA's DIET architecture. We use a typical Rasa pipeline for our experiments on the DIET classifier, which consists of: 1) Tokenization, 2) Featurization, and 3) Entity Recognition/Intent Classification. The Rasa framework allows for a modular approach in creating a model pipeline. We use a Whitespace tokenizer, followed by a set of supervised embedding featurizers, followed by the DIET components. The DIET architecture has two components: intent classification and slot tagging. For intent classification, it captures a representation of the entire utterance by combining individual token representations and passing the result through a transformer layer. For slot tagging, individual token representations obtained from the transformer layer are further fed into a conditional random field (CRF) layer. Finally, the model optimizes on the total loss obtained by combining intent loss and slot loss.

3 Experiments

We evaluate proposed models on two open source dataset ATIS[1] and SNIPS [1]

3.1 Data

Dataset	Train Data	Valid Data	Test Data	Intent	Slot
ATIS	4478	500	893	21	120
SNIPS	13084	700	700	7	72

Table 1: Dataset Information.

Table 1 shows ATIS and SNIPS dataset information.

Airline Travel Information System (ATIS) dataset contains audio recording of people making flight reservations. It contains 4478 training data, 500 validation data and 893 test data. ATIS dataset is highly skewed in nature. In addition, there are 120 slot labels and 21 intent types present. SNIPS dataset is collected from SNIPS personal voice assistant. It contains 13084 training data, 700 validation data and 700 test data. In addition, there are 7 intents and 72 slots. The complexity of SNIPS dataset is high due to large number of cross-domain intents.

3.2 Training Details

3.2.1 Trellis Network Based Experiments

Trellis network has many hyper-parameters, we majorly experimented with different values of number of layers, embedding size and hidden size. We first identified optimal value of number of layers. Bai et al. (2018) used 55 layers for Word-PTB and 70 layers for Word-WT103 datasets for LM task. For joint intent and slot experimentation, we varied the number of layers from 5 to 20. We found that the experimentation provided best accuracy for 11 layers and started decreasing after 11 layers. Hence, all our experimentation consisted of 11 layers.

We kept embedding size small as compare to original LM task, LM experiments were done with embedding size between 280 and 512, whereas we

[1]https://github.com/MiuLab/SlotGated-SLU/tree/master/data

are keeping embedding size 50 & 100 for different experiments. Hidden size of LM was 1000 to 2000, whereas we are keeping 100 or 120 for different experiments. For all experiments, we are keeping nout (output dimension of Trellis network) same as embedding size.

3.2.2 NCIPMA Network

We use glove embedding of 300 dimensional vector for each seen word and "unk" word embedding (randomly initialized 300 dimensional vector) for unseen words to construct weight matrix for Embedding Module. There are three parallel CNN networks. We use Conv1D module with filter size as 1, 2 and 3 respectively and hidden dimension as 256 with padding "same" feature. The inputs for Multi-Head Attention are having same hidden dimension namely 256. Hence, the output dimension is also 256. Addition module adds the output of the three Multi-Head Attention Module. Hence the dimension size is same as 256. We pass through parallel Multi-Head Attention Module, which does not change the hidden dimensional. Hence, the output dimension for each Multi-Head Attention module is 256. For intent, we flatten the matrix to 2D and pass through "Dense" layer, with intent size as hidden units and activation as "Softmax". For slot, we pass through "Dense" layer, with slot size as hidden units and activation as "Softmax".

We use "Keras" platform with optimizer as "Adam", loss as "categorical crossentropy", batch size as 64 and learning rate as 0.001.

3.2.3 NCIPMA with CRF

All the dimensions used for this experiment is same as NCIPMA network except for intent and slot prediction.

For intent prediction, we use CRF layer with output dimensions as intent size, with mode set to join mode. For slot prediction, we use CRF layer with output dimensions as slot size, with mode set to join mode.

We use "Keras" platform with optimizer as "Adam", loss as "crf_loss", accuracy as "crf_viterbi_accuracy", batch size as 64 and learning rate as 0.001.

3.2.4 RASA DIET

For the DIET model, we use the default architecture suggested by RASA for intent classification and slot tagging without pre-trained embeddings. It consists of two transformer layers of size 256, with 4 attention heads. The learning rate is set to 0.001, batch size to 4, and the dropout to 0.2.

4 Results and Analysis

4.1 Impact of embedding and hidden size on Trellis Network

This section explores the impact of embedding and hidden size on Uni-directional and Bi-directional Trellis network.

4.1.1 Unidirectional Trellis Network

Dataset	Emb Size	Hidden Size	Intent Acc	Slot F1 Score
ATIS	50	100	95.0	94.3
ATIS	100	120	**95.33**	**94.44**
SNIPS	50	100	96.89	81.45
SNIPS	100	120	**98.16**	**83.59**

Table 3: Trellis Network results with different model parameters for ATIS and SNIPS.

Table 3 shows results with ATIS and SNIPS data. On both datasets, Accuracy increases little with increase in embedding and hidden sizes.

4.2 Bidirectional Trellis network

Fusion	Emb Size	Hidden Size	Intent Acc	Slot F1 Score
Linear	50	100	**97.88**	**90.01**
Linear	100	120	97.88	88.57
Concat	50	100	96.89	88.75
Concat	100	120	97.31	89.43

Table 4: Bi Directional Trellis Network results for ATIS and SNIPS.

Table 4 shows results of Bidirectional Trellis with SNIPS data set, slot F1 improves a lot as compare to unidirectional model. But with increase in number of parameters, Bi-directional models are not improving well. This might because of limitations of fusion block. Therefore, there is need for trying different fusion techniques with it.

4.3 NCIPMA Architecture Result and Comparison with State of Art

We evaluate all the proposed architectures on open source dataset like ATIS and SNIPS. Table 2 shows the accuracy comparison of the proposed models

	ATIS		SNIPS	
Architecture	**Intent**	**Slot(f1)**	**Intent**	**Slot(f1)**
NCIPMA	97.87	95.42	**98.57**	91.55
NCIPMA with CRF	**97.87**	**96.25**	98.14	92.35
Unidirectional Trellis Network Based Model	95.33	94.44	98.16	83.59
Bi-directional Trellis Network Based Model	95.11	95.70	97.88	90.01
RASA	95.88	94.47	97.56	**92.91**
Goo et al. (2018)	94.1	95.2	97.0	88.8
E et al. (2019)	97.76	95.75	97.29	92.23

Table 2: Accuracy Comparison with State of the Art.

with each other in addition to state of the art models like Slot gated model (Goo et al., 2018) and Bi-directional Interrelated model (E et al., 2019). From the table, we are able to infer that NCIPMA with CRF model is able to surpass state of the art architecture and other architectures for ATIS and SNIPS. For ATIS, intent accuracy improved by 0.11% and slot accuracy improved by 0.5%. For SNIPS, the intent accuracy improved by 0.85% and the slot accuracy improved by 0.12%. NCIPMA architecture without CRF is able to perform better intent detection for SNIPS by 1.28%. We are able to infer that CRF has boosted the accuracy of NCIPMA architecture because final slot prediction is based on previous labels and current word, which aided in improving the slot prediction. In addition, word level intent prediction for ATIS aides in maintaining the accuracy for intent for ATIS and degraded for slot by 0.43%. This indicates the word-level intent evaluation by CRF aids in maintaining the intent accuracy without much degradation. We are also able to infer that CNN with Self-attention architecture is able to beat models with sequential models like GRU, LSTM. RASA for SNIPS slot is performing the best by beating state of the art by 0.68%.

5 Conclusion

We are able to find a replacement for sequential learning models like LSTM, GRU and RNN by using CNN with self-attention. We are able to see that NCIP module is able to project the importance of uni-gram, bi-gram and tri-gram well. Trellis network based models worked well but further research is required with them to improve on intent classification and slot tagging tasks.

Future scopes are exploration of unified model to predict domain, intent and slot for the said task. Exploration of impact of shared weights across layers for CNN with Self-attention is a needed task to reduce size without impact in accuracy.

References

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. Trellis networks for sequence modeling. *arXiv preprint arXiv:1810.06682*.

Haihong E, Peiqing Niu, Zhongfu Chen, and Meina Song. 2019. A novel bi-directional interrelated model for joint intent detection and slot filling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5467–5471.

Mauajama Firdaus, Shobhit Bhatnagar, Asif Ekbal, and Pushpak Bhattacharyya. 2018. Intent detection for spoken language understanding using a deep ensemble model. In *Pacific Rim international conference on artificial intelligence*, pages 629–642. Springer.

Mauajama Firdaus, Ankit Kumar, Asif Ekbal, and Pushpak Bhattacharyya. 2019. A multi-task hierarchical approach for intent detection and slot filling. *Knowledge-Based Systems*, 183:104846.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson HS Liu, Matthew Peters, Michael Schmitz, and Luke S Zettlemoyer. 2017. A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757.

Momchil Hardalov, Ivan Koychev, and Preslav Nakov. 2020. Enriched pre-trained transformers for joint slot filling and intent detection. *arXiv preprint arXiv:2004.14848*.

Joo-Kyung Kim, Gokhan Tur, Asli Celikyilmaz, Bin Cao, and Ye-Yi Wang. 2016. Intent detection using semantically enriched word embeddings. In *2016 IEEE Spoken Language Technology Workshop (SLT)*, pages 414–419. IEEE.

Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. 2016. Leveraging sentence-level information with encoder lstm for semantic slot filling. *arXiv preprint arXiv:1601.01530*.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454*.

Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. 2014. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539.

Chen Tingting, Lin Min, and Li Yanling. 2019. Joint intention detection and semantic slot filling based on blstm and attention. In *2019 IEEE 4th international conference on cloud computing and big data analysis (ICCCBDA)*, pages 690–694. IEEE.

Sevinj Yolchuyeva, Géza Németh, and Bálint Gyires-Tóth. 2020. Self-attention networks for intent detection. *arXiv preprint arXiv:2006.15585*.

Unified Multi Intent Order and Slot Prediction using Selective Learning Propagation

Bharatram Natarajan, Priyank Chhipa, Kritika Yadav and **Divya Verma Gogoi**
research.samsung.com
{bharatram.n, p.chhipa, k.yadav, divya.g}@samsung.com

Abstract

Natural Language Understanding (NLU) involves two important task namely Intent Determination (ID) and Slot Filling (SF). With recent advancements in Intent Determination and Slot Filling tasks, explorations on handling of multiple intent information in a single utterance is increasing to make the NLU more conversation based rather than command execution based. Many has proven this task with huge multi-intent training data. In addition, lots of research have addressed multi intent problem only. The problem of multi intent also pose the challenge of addressing the order of execution of intents found. Hence, we are proposing a unified architecture to address multi intent detection, associated slots detection and order of execution of found intents using low proportion multi-intent corpus in the training data. This architecture consists of Multi Word Importance relation propagator using Multi Head GRU and Importance learner propagator module using self-attention. This architecture has beaten state of the art by 2.58% on MultiIntentData dataset.

1 Introduction

Many voice assistants like Samsung Bixby, Amazon Alexa, Microsoft Cortana, Google Assistant has provided voice solution to ease the phone usage for the users. To make user experience more conversational rather than command oriented, exploration on handling multi intent by NLU is increasing. NLU currently handles three important task for identification. Domain Detector (DD) is the task of identifying which domain or application should execute the utterance. ID is the task of identifying what is the intent of the user from the utterance. SF is the task of identifying the objects of interest (named entities) on which we execute the intent operation. For multi-intent, ID task involves identification of one or more intents in the utterance told by the user. Hence, ID must be able to identify the dynamic number of intents present in the utterance along with identification of the boundaries or segments for each intents. In addition, the order of execution of the identified intents matters as one intent execution might be dependent on the other intent execution. Lastly, we would like to have low proportion of training data for multi–intent so that we reduce the dependency on data generation and maintenance. Hence, we propose a unified architecture that address the following problems: multi intent identification, slot identification, multi-intent boundary detection and execution order of intents.

Lots of work has happened in the area of single intent and slot. Chen et al. (2019) proposes the exploration of BERT architecture for NLU task where pre-trained bi-directional representation on unlabeled corpus, with simple fine tuning, aided in the task of combined single intent prediction and slot detection. E et al. (2019) offer bi-directional interrelated information sharing between intent learnings and slot learnings. In addition, they use new iteration mechanism to enhance the sharing of the learnings. Chen and Yu (2019) project the usage of word attention, calculated using word embedding, in addition to semantic level attention at each decoding step of Bi-LSTM. They also use fusion gate for fusing the intent and slot learnings for enhancing the relationships between intent and slot. Bhasin et al. (2020) recommend the use of Multi modal Bi-Linear Pooling technique for fusing the learnings of intent and slot. Tingting et al. (2019) outlines the usage of Bi-LSTM along with attention for jointly learning the learnings of intent and slot.Xu and Sarikaya (2013a) recommends the usage of convolutional neural network [CNN] along with tri-crf for the joint task of intent detection and slot learning. Liu and Lane (2016a) recommends the usage of attention information along with recur-

Proceedings of the ICON 2020 Workshop on Joint NLP Modelling for Conversational AI, pages 10–18
Patna, India, December 18 - 21, 2020.

rent neural networks in encoder-decoder stage that enhances the learnings of intent and slot. Wang et al. (2018b) recommend the usage of encoder for encoding the sentence representation using CNN , for local feature and high level phrase representation, and Bi-LSTM for capturing contextual semantic information and decoding using attention information, calculated from encoder, in each decoding step of Long Short term Memory [LSTM] decoder. Liu and Lane (2016b) offers the usage of recurrent neural network[RNN] for solving the problem of joint intent and slot by updating the intent detection as and when words are coming from the utterance. Wang et al. (2018a) suggest the usage of bi-model network where two parallel Bi-LSTM are used and they use the hidden information of one Bi-LSTM to another in each network. Then they use the learnings to predict intent and slot from each network. Yu et al. (2018) offer to use cross-attentive information propagation for enhancing the meaning of the word at word level as well as tagging level to aid the task of joint intent and slot prediction. The above networks has explored many ways of capturing important word level information and fusing the learnings for predicting single intent and slot. Inspired on the information captured, explorations on multi intent detection and slots have gathered steam recently to make the task finding generic.

Gangadharaiah and Narayanaswamy (2019) suggest Bi-LSTM encoder for encoding the sentence information and uses sentence level attention information as well as word-level attention information for each time step in both the decoders. One decoder predicts word-level intent detection, another decoder predicts word-level slot detection and one feed forward neural network predicts sentence-level intent prediction. This network suffers from contiguous boundary utterance detection. Xu and Sarikaya (2013b) suggests the usage of share information between multiple intents to identify segments belonging to each intent by using hidden layer to map the learning of word importance to intent prediction. The network is very shallow and is unable to capture the long distance word relationship. Kim et al. (2017) suggest usage of two-stage system to detect multiple intents in a single utterance when the model is trained with single utterance by first breaking the utterance into two chunks in the first stage and processing each chunk sequentially by the model. This method suffers from pipeline approach where error in first stage results in propagative error in model stage. The exploration, in this area, is very less comapred to single intent and slot, dur to the absence of proper open source dataset for multi intent data.Hence, we are proposing a novel architecture, which will address the following: multiple intent detection, intent segmentation or boundaries and execution order of the found multi intents and slot prediction, where execution order determines the relationship between intents to derive sequence of execution on the voice assistant system.All the experiments are run on MultiIntentData dataset, a newly deveoped dataset.

We first address the problem statement in detail in Dataset Section, and then followed by Dataset Pre-Processing for extracting the required information, architecture explanations, results discussion and finally Conclusion.

2 Dataset

We solve four types of problem in this paper namely word-level intent prediction, sentence-level intent prediction, word-level order prediction, and word-level slot prediction. We use word-level intent prediction for finding the boundaries or segments of multiple intents present in the sentence as shown in Figure 1

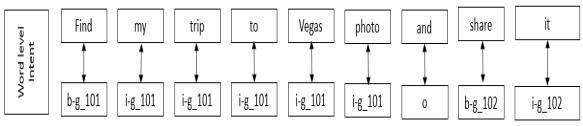

Figure 1: Word level Intent Prediction

We use sentence-level intent prediction for identifying all the intents present in the sentence as shown in Figure 2

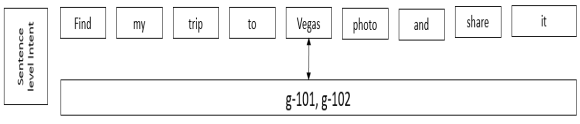

Figure 2: Sentence level Intent Prediction

We use word-level order prediction for finding out the order in which the intent segments or intent boundaries must execute as shown in Figure 3

We use word-level slot prediction for finding all the slots in the sentence as shown in Figure 4.

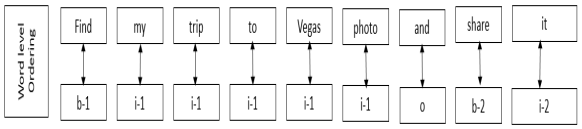

Figure 3: Word level Order Prediction

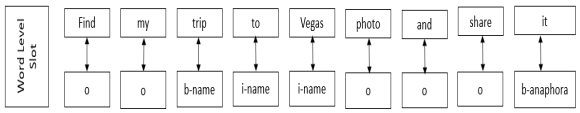

Figure 4: Word level slot prediction

For handling very less multi-intent training data, we have created new training and test dataset with the help of linguist namely MultiIntentData. MultiIntentData dataset was created using single intent information from two domains namely Gallery and Camera.

Gallery domain contains the information shown in Table 1.

Intent ID	Description
g-101	Open gallery with or without using picture name, album name and folder name
g-102	Share the pictures found using picture name, album name and folder name
g-219	Add the pictures found using picture name, album name and folder name to wallpaper

Table 1: Gallery domain information.

Camera domain contains the information shown in Table 2.

Intent ID	Description
c-1	Open camera
c-176	Turn on flash feature in camera
c-23	Change the picture size of front or rear camera and take picture
c-3	Change the modes of camera and take picture
c-408	Create emoji using the taken picture

Table 2: Camera domain information.

The data is created for natural forms of multi-intent voice queries while also ensuring the intent order and dependencies are ensured. For example, we created continuous sentences without any separators between individual intents. Example is "Take the shot in pro mode with the flash". Here the user is requesting to turn on flash in camera and then set the mode to pro before taking picture. In this utterance, there are no separators. Considering the constraints, linguist has created data using four types of combination from single intent data of two domains as shown in Table 3. The created

Combination Type	Example Utterance
Independent intents within domain	share Malibu pictures and add the latest paper to wallpaper
Independent intents across domain	open camera after launching gallery
Dependent intents within domain	find latest Malibu pictures and share it
Dependent intents across domain	take a pic using selfie mode and share it

Table 3: Intent Combinations with example.

data[1] contains 1896 training utterances and 350 test utterances. The training data contains 8% multi-intent training data and 92% single-intent training data using intents from two domains mentioned in Table 1 and Table 2. The test data contains 92% multi-intent data and 8% single intent data.

3 Proposed Method

This section explains Data Preprocessing followed by Model 1, Model 2 and Model 3.

3.1 Data Preprocessing

The training data and test data are present in the format as shown in Figure 5.

<[view my {farewell}(name) album](G_101)>(1) before <[launching the camera](C_1)>(2)

Figure 5: Word level Order Prediction

We write each utterance inside the square brackets followed by intent id inside parenthesis. This represents intent information.

In addition, we write each slot phrase in the utterance by curly brackets followed by slot id inside parenthesis. This represents slot information.

Finally each intent information is written inside the "<" and ">" symbols followed by order id, a number inside parenthesis. This represents order information.

[1]https://github.com/MultiIntentData/MultiIntentData

To get the utterance with intent, slot and order details do the following. First, we extract the order information by using regular expression to search for first encountered "<" and ">" followed by parenthesis. The order information contains intent information inside the "<" and ">" and order id. We extract one or more slot information from intent information by using regular expression to find all the left curly braces and matching right curly braces along with parenthesis. This will give list of (slot phrase, slot id) tuples. Using this list, we generate the original utterance by replacing all the slot information with corresponding slot phrases. Then we use another regular expression to search for left square bracket with matching right square bracket along with parenthesis. This provides (utterance, Intent ID) tuple for intent information. Hence the order tuple becomes ((utterance, Intent ID), Order ID) where the utterance has the intent as Intent ID and order as Order ID. Now we assign Intent Id and order ID for each word in the utterance in IBO format to generate word-level intent information and word-level order information. From the list of (slot phrase, slot ID) tuples , we create IBO format for slots where the phrases, from the utterance, not in the slot phrase are assigned "o" and phrases in slot phrase are assigned Slot ID with first word as "b-Slot ID" and rest of the slot phrases as "i-Slot ID" to generate word-level slot information. We repeat the above steps for another order information within the utterance (If present).

If more than one order is present then there might be phrases not belonging to any order. In such cases, we assign those phrases as "o" for word-level intent, word-level slot and word-level order information. We concatenate the utterances generated in the process. The list of intent ID(s), generated by parsing multiple order information, is assigned as label for the final utterance generated for sentence level intent information.

The next section explains model architecture evolutions.

3.2 Model 1: GRU learner enhancer with Self Attention

Figure 6 shows the architecture of the proposed model. The model is explained in the following subsections.

3.2.1 Utterance Pre-processing

First, we count the number of words in the utterance (W1). If the count is less than max length

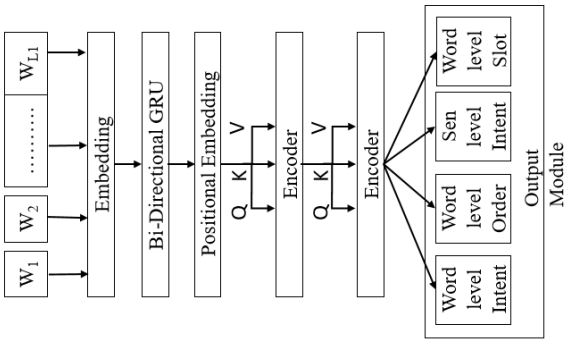

Figure 6: Bi-directional GRU with two Encoders

(L1), then we append the utterance with (W1 –L1) padding words. We use "¡pad¿" symbol as the padding word. Then we index individual word in the utterance using training dictionary. In training dictionary, we assign ¡pad¿ symbol with index 0 and other words (including "unk" word) are indexed one to "N - 1" (N -¿ max number of words in the dictionary). If we do not find the word in the training dictionary, then we assign index of "unk" word. We pass indexed words of the utterance to embedding layer. We use L1 as 23.

3.2.2 Embedding Layer

Embedding layer contains the weight matrix of each index to vector of L2 dimensions. Its dimension is N * L2. We pass each indexed word through the weight matrix to get its vector of L2 dimensions. Since there are L1 words present in the utterance, we get L1 * L2 matrix. Then we pass this matrix to single GRU unit. We use glove embedding of size 300 dimension to map word index to its glove-embedding vector of 300 dimension. We assign "unk" word with random initialization of 300-dimension vector. Hence L2 is 300.

3.2.3 Bi-directional GRU Layer

Gated Recurrent Unit (GRU) layer is a gated mechanism, where it uses reset gate and update gate for information propagation at each time step.

The update gate decides what information needs to propagate by passing previous hidden state and current input through sigmoid function.

Sigmoid function squashes the value between zero and one. If the value is closer to zero then we do not propagate the info. If the value is closer to one we propagate the info.

The reset gate decides what information from the past needs to propagate by passing the previous hidden state and current input through sigmoid

function and multiplying the output with previous hidden state. The output, calculated using sigmoid function, contains what value needs to stay and what value needs to forget. Multiplying the output with previous hidden state updates the past information propagation as shown in Equation 1.

$$\begin{aligned} z_t &= \sigma_g(W_z * x_t + U_z * h_{t-1} + b_z) \\ r_t &= \sigma_g(W_r * x_t + U_r * h_{t-1} + b_r) \\ h_t &= z_t * h_{t-1} + (1-z_t)* \\ &\quad \phi_h(W_h X_t + U_h(r_t * h_{t-1}) + b_h) \end{aligned} \tag{1}$$

where z_t represent update gate and r_t represent reset gate. We use bi-directional GRU where we concatenate the outputs of forward GRU and backward GRU.

We pass the concatenated output of Bi-directional GRU after adding with trigonometric position embedding. Trigonometric positional embedding generates alternate sine and cosine embedding taking position to generate embedding. We pass the combined embedding to transformer encoder module. We use 256 as hidden dimension of Bi-directional GRU.

3.2.4 Transformer Encoder

We use transformer encoder module (Devlin et al., 2018) using multi-head attention layer, positional feed forward layer, and residual connection layer followed by normalization. We use two encoder modules.

Multi Head attention network splits the input embedding into "n" equal chunks and we provide each chunk as input to self-attention. Self-attention layer aides in enhancement of word importance over the entire sentence. We achieve this by providing the encoded input representation as a set of key-value pair. Then we provide query as same encoded input sentence. We use scalar dot product attention where we apply dot product between query and all the keys to provide weighted sum and then we multiply with value to provide weighted sum of the value. We concatenate the "n" self-attention outputs. This output contains the information importance from different subspaces from different positions. We pass this output to residual connection module. We use "n" as 16. This module produces 256 as hidden dimension output.

Residual connection module takes the input and output of multi-head attention module as input to this module and apply element wise addition on this module. We give the output to Normalization layer. This module produces 256 as hidden dimension output.

Normalization layer apply normalization on the input layers. We provide the output to position wise feed forward neural network. This module produces 256 as hidden dimension output.

Position wise feed forward neural network enhance the learning of the word level importance. This module produces 256 as hidden dimension output.

We pass the output of Positional Feed Forward neural network as input to another encoder and repeat above steps. The output of the second encoder contains 256 as hidden dimension.

We finally pass the output of second Positional Feed forward neural network to output module.

3.2.5 Output module

The module consist of four fully connected feed forward neural networks. Each feed forward network module predicts word level slot, word level intent, word level order and sentence level intent using softmax on the first three networks and sigmoid on the last one. The first three provides multi class classification and last one provides multi label classification. There are 17 word-level intent, 5 word-level order, 20 word-level slot and 8 sentence-level intents to identify.

3.2.6 Analysis

The proposed Model-1 able to handle simple relationship between the intents and orders.

However, it is not able to capture the relationship between intent and order boundaries properly. Increase in hidden dimension leads to poor performance of the model due to less training data. In addition, the model is not able to identify the boundaries of slot properly. Finally, the model is not able to differentiate between the presence of word as part of separator and presence of word as part of open title type slot.

Consider the example "display camera app to make me a crazy emoji". In this, the word-level intent prediction segments the sentence properly.

Whereas the word-level order prediction is not segmenting the sentence properly. It predicts only one order when the intent is clearly presenting two intents as shown in Figure 7 and Figure 8.

This clearly shows that common module is not sufficient to use both intent as well as order prediction. The fact that the intent and order predictions

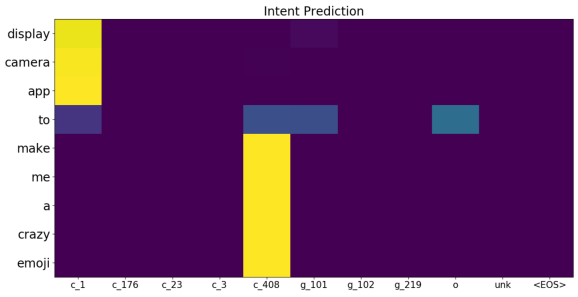

Figure 7: Intent Prediction. Brighter color indicates stronger relationship

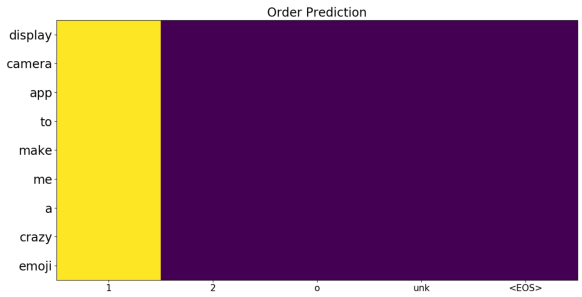

Figure 8: Order Prediction. Brighter color indicates stronger relationship

are inter-related propagated the idea of Model 2 explained in the next section.

3.3 Model 2 : Multi Head GRU with Selective Learning Propagation block

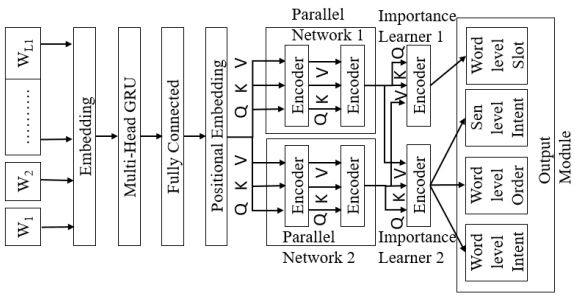

Figure 9: Multi Head GRU with Selective Learning Propagation

Figure 9 shows the architecture of the proposed model.

First, we process the utterance as explained in 3.2.1 section. Then we pass the output, indexed words of length L1, to embedding layer. Embedding layer process the indexed utterance and generates the embedding vector (L2) for each word as explained in 3.2.2 section. Now we pass this to Multi-Head GRU module, which we explain in the next section

3.3.1 Multi Head GRU

We split the input embedding into four equal chunks. We give each chunk to one bi-directional GRU. Bi-directional GRU provides contextual information of the current word with respect to future data and past data. We explained this in 3.2.3 section. We concatenate the output of each parallel Bi-directional GRU. The hypothesis behind Multi Head GRU is we capture different phrase importance from different positions. Then we pass through fully connected network to pick the important phrases from concatenated output. We sent the fully connected network output to parallel network, which we explain in the next section. We use 256 as hidden dimensional information.

3.3.2 Parallel Network modules

The module has two parallel transformer encoder block. We have explained the working of Transformer Encoder block in 3.2.4 section. The reason for two parallel networks is different learning representation is available for the same input. In addition, we use one network for predicting the intent related predictions and other for slot related predictions. This makes each network learnings to concentrate on related task only thereby dividing the learnings of the task between the two networks. Now we selectively propagate the learnings of each network for each task using Learner module, which we explain, in the next section.

3.3.3 Importance Learner Module

This module is the most important module. It takes the output of two parallel network modules and selects the information from one parallel network module to enhance the learning of another parallel network module. We achieve this using the self-attention block in transformer encoder module where we use query and key as the output of one parallel network module and value as other parallel network module instead of passing the same input as query, key and value. We have explained the working of self-attention block in 3.2.4 section. Since there are two networks that requires learning enhancement we use two learning module. Importance Learner 1 takes query and key as output of parallel network module 2 and value as output of parallel network module 1. Importance Learner 2 takes query and key as output of parallel network module 1 and value as output of parallel network module 2. We provide the output of Importance Learner 2 to Output module to predict word-level

intent, word-level order and sentence-level intent. Similarly, we provide the output of Importance Learner 1 to output module to predict word-level slot.

Output module takes the output of Importance Learner 1 and Importance Learner 2 and does final prediction. We have explained the working of Output module in 3.2.5 section.

3.3.4 Analysis

The model is able to differentiate the boundaries of open title kind of slots properly. In addition, the model is able to understand the difference between the words being part of open title slot and the words acting as separator.

There is improvement in the relationship understanding between intent and order boundaries. However, the model is suffering from order mix-up within the boundary. However, the model is suffering from order mix-up within the boundary. In addition, confusion point exist between open title slots. Consider the example "click photo in food mode after turning on flash". In this, the model is able to find the intent boundaries properly but the order boundaries is not proper due to relationship misunderstanding between "click photo" and "after" as shown in Figure 10 and Figure 11.

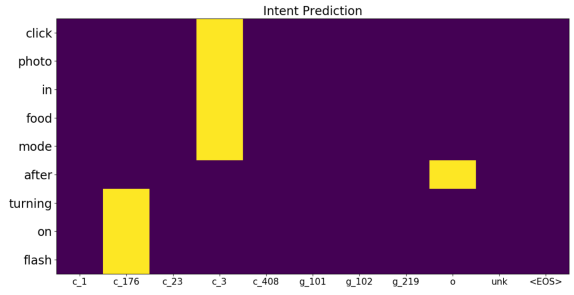

Figure 10: Intent Prediction. Brighter color indicates stronger relationship

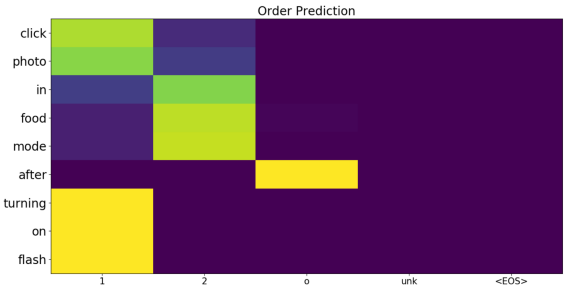

Figure 11: Order Prediction. Brighter color indicates stronger relationship

The limitations clearly shows the need for new way of addressing the problem of intent and order boundary relationship as well as improvement in open title slot detection.

3.4 Model 3: Multi Head GRU with Importance Learner Module and Order processing

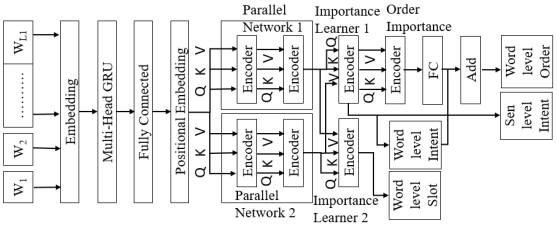

Figure 12: Multi Head GRU with Importance Learner module and Order Importance Module

Figure 12 shows the proposed architecture of Model 3.

We pre-process the input utterance as mentioned in 3.2.1 section. We get output as indexed words for the utterance whose count will be L1.

We pass the utterance containing indexed words of length L1 to embedding layer. Embedding layer assigns vector to each word as explained in 3.2.2 section. The output will be a matrix of L1 words with each word having embedding vector of length L2. Hence its dimension will be L1 * L2. We pass the matrix to Parallel network modules.

Parallel network modules generate different representational information for the same input as well as information learning is divided between the networks as explained in 3.3.2 section. Both the outputs are provided as input to Importance Learner module.

Importance Learner module selectively chooses the learning of one network to influence the learning of other network as explained in 3.3.3 module. Since there are two networks, we use two Importance Learner modules separately to enhance each network learnings. The output of Importance Learner 1 is given to Order Importance learner module as explained in the next section. We also use the output to predict word level intent and sentence-level intent.

3.4.1 Order Importance learner module

This module enhances the learning by passing through another transformer encoder layer where we use the same input as query, key and value. The working of transformer encoder is explained

Architecture	Slot Sentence level Acc	Word Intent Sentence level Acc	Sen Intent Sentence level Acc	Word Order Sentence level Acc	Overall Sentence Level Acc
Model 1	89.14	88	88.57	86.57	86.57
Model 2	91.43	88.29	88.86	89.14	88.29
Model 3	**94.86**	**90.29**	**90.29**	**90.29**	**90.29**
Gangadharaiah and Narayanaswamy (2019)	90.86	89.14	89.71	87.71	87.71

Table 4: Comparison of state of the art models.

in 3.2.4 section.We add the module output with word-level intent output. For this, we reduce the hidden dimension of the network to intent number by passing through intent fully connected network with "relu" activation.

The Order Importance Learner module output predict word-level order by passing through fully connected layer with softmax as activation.

The Importance Learner 2 predicts word-level slot.

3.4.2 Analysis

The model, when ran on MultiIntentData dataset, is able to differentiate the open title slots well. In addition, the slot boundaries have improved. In addition, the word differentiation as part of open title or part of separator is able to identify properly. In addition, we are able to see huge improvement in intent and order boundary understanding. Finally, the model has reduced the confusion within order boundaries.

There is a need of improvement in better intent detection and slot detection for few cases.

4 Result and Discussion

We ran all the three models using MultiIntentData dataset. We modified Rashmi and Narayanaswamy (2019) architecture to predict order as well, similar to prediction of word-level intent by adding new decoder and providing same attention information for each decoder step to predict word-level order, and ran on MultiIntentData dataset to compare with state of the art. All the result are captured in Table 4.

From the table we are able to beat the state of the art architecture by 2.58%. This is attributed to the fact that parallel network along with importance learner module is able to enhance the learning of intent, slot and order when compared to unified architecture proposed in Gangadharaiah and Narayanaswamy (2019). In addition, the word differentiation between part of the catchall and part of the separator is handled well by Model 3.

From the result in Table 4, we are able to understand that Model 3 is the best performing model over Model 1 and Model 2 by 3.72% and 2% respectively. From the result, we are able to see that Model 3 is able to perform open slot distinction i.e. distinction between the slots, open slot boundary detection and word boundary detection between part of open slot and part of separator over Model 1. For more details please have a look at 3.2.6 and 3.4.2. Model 3 is able to improve the order and intent boundaries well as well as confusion points between open title slots. For more details, please see 3.3.4 and 3.4.2.

5 Conclusion

This work showed the importance of multi intent detection with associated slots and its order of execution over single intent and slot. This also showed the importance of multi intent learning using low corpus data. We are able to derive that Multi Head GRU aide in better contextual understanding of the input embedding representation. In addition, the presence of parallel network for intent and slot learning, along with two-importance learner module has shown better understanding on the differentiation between boundaries of the slot and start of the next intent. In addition, the word level intent aided in influencing the overall sentence level intent. The word level intent as well as the separator also influence the intent ordering execution. This has resulted in improvement to the tune of 3.72%. Multi head self-attention learning on context aided in improving from state of the art model by 2.58%.

Future scope is to resolve anaphora resolution of slots within intent and across intent.

References

Anmol Bhasin, Bharatram Natarajan, Gaurav Mathur, and Himanshu Mangla. 2020. Parallel intent and slot prediction using mlb fusion. In *2020 IEEE 14th International Conference on Semantic Computing (ICSC)*, pages 217–220. IEEE.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*.

Sixuan Chen and Shuai Yu. 2019. Wais: Word attention for joint intent detection and slot filling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9927–9928.

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

Haihong E, Peiqing Niu, Zhongfu Chen, and Meina Song. 2019. A novel bi-directional interrelated model for joint intent detection and slot filling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5467–5471.

Rashmi Gangadharaiah and Balakrishnan Narayanaswamy. 2019. Joint multiple intent detection and slot labeling for goal-oriented dialog. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 564–569.

Byeongchang Kim, Seonghan Ryu, and Gary Geunbae Lee. 2017. Two-stage multi-intent detection for spoken language understanding. *Multimedia Tools and Applications*, 76(9):11377–11390.

Bing Liu and Ian Lane. 2016a. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454*.

Bing Liu and Ian Lane. 2016b. Joint online spoken language understanding and language modeling with recurrent neural networks. *arXiv preprint arXiv:1609.01462*.

Chen Tingting, Lin Min, and Li Yanling. 2019. Joint intention detection and semantic slot filling based on blstm and attention. In *2019 IEEE 4th international conference on cloud computing and big data analysis (ICCCBDA)*, pages 690–694. IEEE.

Yu Wang, Yilin Shen, and Hongxia Jin. 2018a. A bi-model based rnn semantic frame parsing model for intent detection and slot filling. *arXiv preprint arXiv:1812.10235*.

Yufan Wang, Li Tang, and Tingting He. 2018b. Attention-based cnn-blstm networks for joint intent detection and slot filling. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, pages 250–261. Springer.

Puyang Xu and Ruhi Sarikaya. 2013a. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 78–83.

Puyang Xu and Ruhi Sarikaya. 2013b. Exploiting shared information for multi-intent natural language sentence classification. In *INTERSPEECH*, pages 3785–3789.

Shuai Yu, Lei Shen, Pengcheng Zhu, and Jiansong Chen. 2018. Acjis: A novel attentive cross approach for joint intent detection and slot filling. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE.

EmpLite: A Lightweight Sequence Labeling Model for Emphasis Selection of Short Texts

Vibhav Agarwal, Sourav Ghosh, Kranti Chalamalasetti, Bharath Challa,
Sonal Kumari, Harshavardhana, Barath Raj Kandur Raja

{vibhav.a, sourav.ghosh, kranti.ch, bharath.c, sonal.kumari,
harsha.vp, barathraj.kr}@samsung.com

Samsung R&D Institute Bangalore, Karnataka, India 560037

Abstract

Word emphasis in textual content aims at conveying the desired intention by changing the size, color, typeface, style (bold, italic, etc.), and other typographical features. The emphasized words are extremely helpful in drawing the readers' attention to specific information that the authors wish to emphasize. However, performing such emphasis using a soft keyboard for social media interactions is time-consuming and has an associated learning curve. In this paper, we propose a novel approach to automate the emphasis word detection on short written texts. To the best of our knowledge, this work presents the first lightweight deep learning approach for smartphone deployment of emphasis selection. Experimental results show that our approach achieves comparable accuracy at a much lower model size than existing models. Our best lightweight model has a memory footprint of 2.82 MB with a matching score of 0.716 on SemEval-2020 (shallowLearner, 2020) public benchmark dataset.

Index terms: emphasis selection, mobile devices, natural language processing, on-device inferencing, deep learning.

1 Introduction

Emphasizing words or phrases is commonly performed to drive a point strongly and/or to highlight the key terms and phrases. While speaking, speakers can use tone, pitch, pause, repetition, etc. to highlight the core of a speech in the minds of an audience. Similarly, while writing or messaging, authors can emphasize the words by customizing the formatting like typeface, font size, bold, italic, font color, etc. as illustrated in Figure 1. With the explosion of social media and messaging platforms, word emphasis has become more critical in engaging readers' attention and conveying the author's message in the shortest possible time.

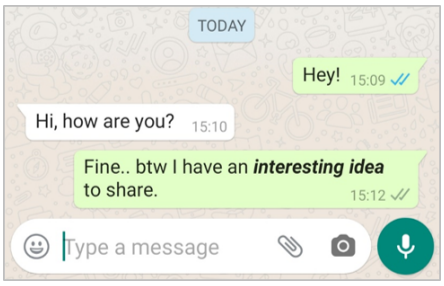

Figure 1: Prominent words in a message are being emphasized (**Bold** + *Italic*) automatically

Emphasis selection of text has recently emerged as a focus of research interest in natural language processing (NLP). The goal of emphasis selection is to automate the identification of words or phrases that bring clarity and convey the desired meaning. Automatic emphasis selection can help in better graphic designing and presentation applications, as well as can enable voice assistants and digital avatars to realize expressive text-to-speech (TTS) synthesis. High-quality emphasis selection models can enable automatic design assistance for creating flyers, posters and accelerate the workflow of design programs such as Adobe Spark (Adobe, 2016), Microsoft PowerPoint, etc. These emphasis selection models can also empower digital avatars like Samsung Neon (NEON, 2020) to achieve human-like TTS systems. Understanding emphasis selection is also crucial for many downstream applications in NLP tasks including text summarization, text categorization, information retrieval, and opinion mining.

Proceedings of the ICON 2020 Workshop on Joint NLP Modelling for Conversational AI, pages 19–26
Patna, India, December 18 - 21, 2020.

In the current work, we propose a novel lightweight neural architecture for automatic emphasis selection in short texts, which can perform inference in a low-resource constrained environment like a smartphone. Our proposed architecture achieves near-SOTA performance, with as low as 0.6% of its model size.

2 Related Work

Prior work in NLP literature towards identifying important words or phrases have focused widely on *keyword or key-phrase extraction*. Considerable progress has been made in keyword or key-phrase extraction systems for long documents such as news articles, scientific publications, etc. (Rose et al., 2010). The core operation procedure of these systems is to extract the nouns and noun phrases. To achieve these, researchers have used statistical co-occurrence (Matsuo and Ishizuka, 2004), SVM (Zhang et al., 2006), CRF (Zhang, 2008), graph-based extraction (Litvak and Last, 2008), etc. Recent efforts have even expanded the idea from a set of documents to social big data (Kim, 2020). However, in the context of short texts like text messages, headlines, or quotes, keyword extraction systems often mislabel most nouns as important without considering the essence of the text, thus performing poorly at the task.

Emphasis selection aims to overcome this by scoring words which properly capture the essence of a text by focusing on subtle cues of emotions, clarifications, and words that capture readers' attention, as seen in Table 1. Recent research interest towards these tasks often uses label distribution learning (Shirani et al., 2019). MIDAS (Anand et al., 2020) uses label distribution as well as contextual embeddings. One drawback of using label distribution learning is the requirement of annotations, which are not readily available in most datasets. Pre-trained language model has also been used to achieve emphasis selection (Huang et al., 2020). Singhal et al. (Singhal et al., 2020) achieves significantly good performance with (a) Bi-LSTM + Attention approach, and (b) Transformers approach. To achieve their modest performances, these architectures produce huge models. For instance, IITK model (Singhal et al., 2020) takes up 469.20 MB in BiLSTM + Attention approach, while requiring almost 1.5 GB in Transformers approach. This is partly due to the use of embeddings like BERT (1.2 GB) (Devlin et al., 2018), XLNET (1.34 GB) (Yang et al., 2019), RoBERTa (1.3 GB) (Liu et al., 2019), etc. General-purpose models that emphasize on model size still consume significant ROM: 200 MB (for DistilBERT (Sanh et al., 2019)) and 119 MB (for MobileBERT (Sun et al., 2020b) quantized int8 saved model and variables; sequence length 384). Thus, in spite of the performance benefits, these emphasis selection systems with high-memory footprints are not suitable for the storage specifications of mobile devices.

Table 1: Keyword Extraction (MonkeyLearn, 2020) vs. Emphasis Selection

Input Text	**Keywords/Key phrases Detected**	**Emphasis Selection**
A simple I love you means more than money	A simple I love you means more than **money**	A simple **I love you** means more than money
Traveling – It leaves you speechless then turns you into story teller	Traveling – It leaves you **speechless** then turns you into **story teller**	**Traveling** – It leaves you **speechless** then turns you into **story teller**
Challenges are what make life more interesting and overcoming them is what makes life meaningful	**Challenges** are what make life more **interesting** and overcoming them is what makes life **meaningful**	**Challenges** are what make life more **interesting** and **overcoming** them is what makes life **meaningful**

Thus, while keyword extraction systems are not suitable for short text content, emphasis selection systems perform much better at such tasks. However, most existing architectures of the latter are not light-weight, and thus, not suitable for on-device inferencing on low-resource devices. This motivates us to propose EmpLite, which (a) outperforms keyword extraction systems by using emphasis selection for use with short texts, and (b) differs from existing emphasis selection architectures by ensuring a very light-weight model for efficient on-device inferencing on mobile devices. Our decisions towards achieving low model size include using a subset of GloVe (Pennington et al., 2014) word embeddings, thus, reducing embedding size from 347.1 MB to 2.5 MB, which we discuss in section 4.1.

Table 2: A short text example from dataset along with its nine annotations

Word	A1	A2	A3	A4	A5	A6	A7	A8	A9	Freq [B|I|O]	Emphasis Prob (B+I)/(B+I+O)
Kindness	B	B	B	O	O	O	B	B	B	6|0|3	0.666
is	O	O	O	O	O	O	I	I	O	0|2|7	0.222
like	O	O	O	O	O	O	I	I	O	0|2|7	0.222
snow	O	O	B	O	O	O	I	I	O	1|2|6	0.333

3 Data Collection

We use the officially released SemEval-2020 dataset (RiTUAL-UH, 2020), which is the combination of Spark dataset (Adobe, 2016) and Wisdom Quotes dataset (Quotes, 2020). The dataset consists of 3,134 samples labeled for token-level emphasis by multiple annotators. There are 7,550 tokens with fewer than 10 words in a sample and they are randomly divided into training (70%), development (10%), and test (20%) sets by the organizers. Table 2 shows a short text example from the training set, annotated with the BIO annotations, where 'B (beginning) / I (inside)' and 'O (outside)' represent emphasis and non-emphasis words, respectively, as decided by an annotator. The last column shows the emphasis probability for a word, computed as (B+I) divided by the total number of annotators, i.e. 9. We generate data labels for model training using emphasis probabilities by assigning 0 to low emphasis words (having probability $<$ threshold$_{prob}$) and 1 to high emphasis words (probability $\geq$ threshold$_{prob}$). We experiment with different values for threshold$_{prob}$ and observe that 0.4 yields the best results.

3.1 Evaluation Metric

The evaluation metric for our problem is defined as follows:

Match$_m$ (shallowLearner, 2020): For each instance x in the test set D_{test}, we select a set $S_m^{(x)}$ of $m \in (1..4)$ words with the top m probabilities with high emphasis according to the ground truth. Analogously, we select a prediction set $\hat{S}_m^{(x)}$ for each m, on the basis of the predicted probabilities. We define matching score, or Match$_m$, as:

$$\text{Match}_m = \frac{\sum_{x \in D_{\text{test}}} \left| S_m^{(x)} \cap \hat{S}_m^{(x)} \right| \Big/ m}{|D_{\text{test}}|} \quad (1)$$

Then, we compute the average rank score by averaging all possible Match$_m$ scores:

$$\text{Average Score} = \frac{\sum_{m \in (1..4)} \text{Match}_m}{4} \quad (2)$$

3.2 Data Augmentation

There are only 3,134 annotated samples in the training data, which makes it difficult to improve the accuracy with our neural model. So, to enlarge the amount of training data, we experiment with four data augmentation strategies (Sun et al., 2020a):

1. Randomly removing ≤ 1 word per sentence,
2. Randomly removing ≥ 1 word per sentence,
3. Upper-casing a word randomly, and
4. Reversing the sentence.

The effect of these techniques on our accuracy metric is presented in Section 5.1.

4 System Overview

We begin with a basic model and enhance that model with contextual information (in the form of pre-trained embeddings, char-level embeddings, Parts of Speech Tag concatenation, etc.). We describe the key components (layers) of our final EmpLite neural network architecture, as illustrated in Figure 2.

4.1 Character and Word level features

A combination of word-level and character-level input representations has shown great success for several NLP tasks (Liang et al., 2017). This is because word representation is suitable for relation classification, but it does not perform well on short, informal, conversational texts, whereas char representation handles such informal texts very well. To take the

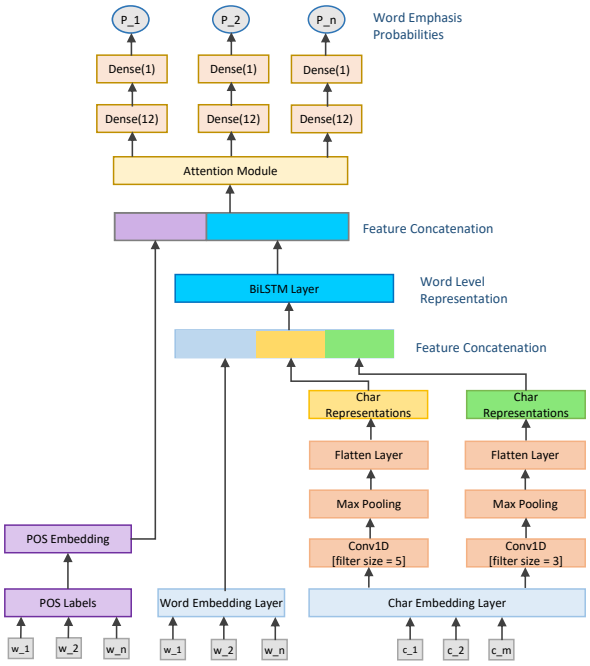

Figure 2: The proposed EmpLite Model Architecture

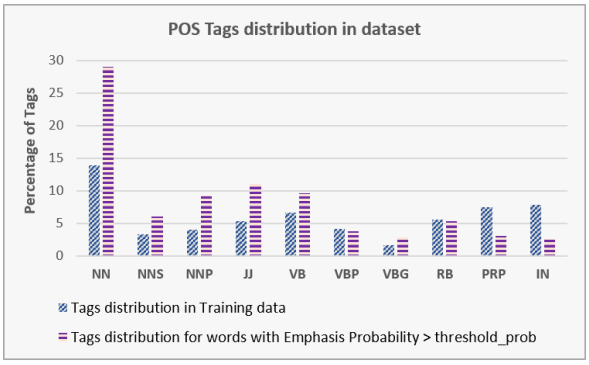

Figure 3: Percentage distribution of top POS tags in training data and for words with emphasis probability greater than threshold

best of both representations, our proposed EmpLite model employs a combination of word and character encodings for a robust understanding of context.

We used two layers of CNN (Ma and Hovy, 2016) with 1D convolutional layers of filter sizes 3 and 5 that extracts multiple character-level representations and handles misspelled words as well as models sub-word structures such as prefixes and suffixes. We use the same number of filters for both convolutional layers: 16, selected on the basis of experimental results for optimal accuracy.

The character level embeddings obtained are then concatenated with pre-trained GloVe (Pennington et al., 2014) 50-dimensional word embeddings. Here, we use a subset of the GloVe embeddings corresponding to training set vocabulary (4331 words), bringing the embedding size down to 2.5 MB. We do not use ELMo (Peters et al., 2018) or other deep contextual embeddings, as it is not feasible to port these heavy pre-trained models for on-device inferencing. In order to handle words not part of training vocabulary, we use a representation, `<UNK>` token. We set the word embedding layer as trainable as that yields the best score due to fine-tuning of layer weights on our task. The i^{th} word encoding, o_{w_i}, is computed as:

$$o_{w_i} = \text{concat}\left(e_{w_i}, \text{CNN}_1\left(e_{c_1}, e_{c_2}, ..., e_{c_n}\right), \text{CNN}_2\left(e_{c_1}, e_{c_2}, ..., e_{c_n}\right)\right) \quad (3)$$

where, e_{w_i} is the word embedding for each word, w_i, in the dataset and e_{c_i} is the character embedding for the input character c_i.

4.2 Word level BiLSTM

The concatenated word representations obtained are then passed through a BiLSTM (Hochreiter and Schmidhuber, 1997) layer with 16 units. The BiLSTM layer extracts the features from both forward and backward directions and concatenates the output vectors from each direction. Also, regular and recurrent dropouts with value 0.2 are applied to reduce model overfitting. Let $\overrightarrow{r}$ and $\overleftarrow{r}$ be the forward and backward output states of the BiLSTM. Then, the output vector, r_b, is defined as:

$$r_b = \overrightarrow{r} \oplus \overleftarrow{r} \quad (4)$$

4.3 Part of Speech (POS) Tag feature

Figure 3 illustrates occurrence of top 10 POS Tags (Marcus et al., 1994) in our training data. We can infer that POS acts as an important input modeling feature as words with POS Tag: Noun (NN, NNP, NNS), Adjective (JJ) or Verb (VB, VBP, VBG) usually have high emphasis probability whereas Pronouns (PRP) and Prepositions (IN) are less likely to be emphasized. Therefore, we use 16-dimensional embedding to encode POS tag information,

Table 3: Comparison of different model architectures

Model	Model Size (MB)	Match_m				Average Score
		m = 1	**m = 2**	**m = 3**	**m = 4**	
Base: Word_Emb + BiLSTM + Dense Layer	1.10	0.479	0.639	0.731	0.785	0.659
Concat[**LSTM(Char_Emb)** + Word_Emb] + BiLSTM + Dense Layer	1.10	0.473	0.658	0.739	0.786	0.664
Concat[LSTM(Char_Emb) + **Word_GloVe** (Non-trainable)] + BiLSTM + Dense Layer	1.02	0.514	0.660	0.748	0.795	0.679
Concat[**CNN**(Char_Emb) + Word_GloVe (Non-trainable)] + BiLSTM + Dense Layer	1.04	0.523	0.669	0.754	0.801	0.687
Concat[CNN(Char_Emb) + Word_GloVe (**Trainable**)] + BiLSTM + Dense Layer	2.70	0.538	0.680	0.766	0.811	0.699
Concat[CNN_1 (Char_Emb) + $\textbf{CNN}_2$ **(Char_Emb)** + Word_GloVe (Trainable)] + BiLSTM + Dense Layer	2.77	0.528	0.690	0.771	0.810	0.701
Above Model + **Attention**	2.80	**0.549**	0.684	0.779	0.817	0.707
EmpLite: Above Model + **POS Feature** Concatenation	2.82	0.541	**0.698**	**0.782**	**0.823**	**0.711**

which is concatenated with the output of the Bi-LSTM layer (obtained from Equation 4):

$$\overrightarrow{h} = \text{concat}\,(r_b, e_{pos}) \tag{5}$$

where, e_{pos} is the POS feature embedding for the sequence.

4.4 Attention Layer

We add the attention (Vaswani et al., 2017) layer to effectively capture prominent words in the input text sequence. The attention weight is computed as the weighted sum of the output of the previous layer, as shown below:

$$Z = \text{softmax}\left(w^T\left(\tanh\left(\overrightarrow{h_1}, \overrightarrow{h_2}, ..., \overrightarrow{h_i}, ..., \overrightarrow{h_n}\right)\right)\right) \tag{6}$$

where, $\overrightarrow{h_i}$ represents output vector of the previous layer, and w^T is the transpose of the trained parameter vector.

The attention layer output is then passed through two time-distributed dense layers with 12 and 1 units, respectively, with sigmoid activation function to output emphasis probability with respect to each word.

5 Experimental Settings & Results

We attempt numerous small changes to our model to enhance the performance. We choose the hyperparameters to optimize accuracy while maintaining a small model size. As the proposed solution is for mobile devices, we have also captured a system-specific metric, the model size in MB. We use the Tensorflow framework (Abadi et al., 2016) for building the models. Table 3 shows the comparison of Match_m scores across different variants of lightweight models evaluated on test data.

The total number of trainable parameters vary in the range of 21,574 to 238,620 for all the model results reported in Table 3. We train the models with 32 batch-size and compile the model using Adam optimizer (Kingma and Ba, 2014). We observe that using CNN gives a better score as compared to LSTM because varying the size of kernels (3 and 5) and concatenating their outputs allow the model to detect patterns of multiple sizes.

We can also infer that using 50-dimensional GloVe embeddings and setting it as trainable improves the overall matching score. This is because we are utilizing language semantic knowledge acquired from the pre-trained embeddings and then fine-tuning it for our task. However, there is an increase in model size due to more number of trainable parameters. Furthermore, we observe marginal gains in the matching score by adding POS tag as a feature followed by an attention layer as these help in a better identification of prominent keywords based on the context.

Figure 4: Emphasis Heatmap for test set samples with word probabilities from EmpLite

Table 4: Comparison with SOTA (Singhal et al., 2020)

Model	Match_m	Size (MB)
IITK: BiLSTM + Attention Approach	0.747	469.20
IITK: Transformers Approach	**0.804**	1536.00
EmpLite	0.716	**2.82**

Figure 4 presents the Emphasis Heatmap for some examples from the test set using our final model. In Table 4 we benchmark our EmpLite model with the state-of-the-art solution by IITK (Singhal et al., 2020) which utilized huge pre-trained models like ELMo, BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and XLNet (Yang et al., 2019). These models require huge RAM/ROM for on-device inferencing making it unsuitable for edge devices where resources are constrained.

Table 5: Data Augmentation Analysis

Augmentation Approach	Dataset modified (%)	Match_m Score
None	0	0.711
Word removal (≤1 per sentence)	20	0.705
	50	0.702
	100	**0.716**
Word removal (≥1 per sentence)	20	0.711
	50	0.704
	60	0.712
	70	0.705
Upper-casing a word	30	0.687
Reversing the sentence	10	0.704
	100	0.707

5.1 Data Augmentation Analysis

Table 5 shows that there is a little score gain by applying data augmentation techniques. For each strategy, we experiment by applying the augmentation approach to different percentages of the total training data and calculated Match_m score. We observe that word upper-casing strategy results in a significant drop in the score due to model overfitting whereas word removal strategy (maximum 1 word per sentence) on entire training data gives highest Match_m score of 0.716.

6 Conclusion

Modeling lightweight neural models that can run on low-resource devices can greatly enhance the end-user experience. In this work, we propose a novel, lightweight EmpLite model for text emphasis selection that can run on edge devices such as smartphones for choosing prominent words from short, informal text. We approach the emphasis selection problem as a sequence labeling task and multiple experiments have shown consistent improvement in the accuracy. Our experimental results show the impact of the attention layer and of using POS as an additional feature in boosting the matching score. Our best performing model achieves an overall matching score of 0.716 with a size of 2.82 MB, proving its effectiveness to run on low-resource edge devices.

Future work includes increasing the vocabulary with commonly used words in English and exploring thin versions of BERT like DistilBERT (Sanh et al., 2019), MobileBERT (Sun et al., 2020b), and TinyBERT (Jiao et al., 2020) for modeling emphasis.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale ma-

chine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Adobe. 2016. Adobe Spark. https://spark.adobe.com, accessed 2020-11-28.

Sarthak Anand, Pradyumna Gupta, Hemant Yadav, Debanjan Mahata, Rakesh Gosangi, Haimin Zhang, and Rajiv Ratn Shah. 2020. Midas at semeval-2020 task 10: Emphasis selection using label distribution learning and contextual embeddings. *arXiv preprint arXiv:2009.02619*.

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

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Zhengjie Huang, Shikun Feng, Weiyue Su, Xuyi Chen, Shuohuan Wang, Jiaxiang Liu, Xuan Ouyang, and Yu Sun. 2020. Ernie at semeval-2020 task 10: Learning word emphasis selection by pre-trained language model. *arXiv preprint arXiv:2009.03706*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Hyeon Gyu Kim. 2020. Efficient keyword extraction from social big data based on cohesion scoring. *Journal of the Korea Society of Computer and Information*, 25(10):87–94.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Dongyun Liang, Weiran Xu, and Yinge Zhao. 2017. Combining word-level and character-level representations for relation classification of informal text. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 43–47.

Marina Litvak and Mark Last. 2008. Graph-based keyword extraction for single-document summarization. In *Coling 2008: Proceedings of the workshop Multi-source Multilingual Information Extraction and Summarization*, pages 17–24.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, page 114–119, USA. Association for Computational Linguistics.

Yutaka Matsuo and Mitsuru Ishizuka. 2004. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(01):157–169.

MonkeyLearn. 2020. Keyword Extraction: A Guide to Finding Keywords in Text. https://monkeylearn.com/keyword-extraction/, accessed 2020-07-03.

NEON. 2020. NEON. https://www.neon.life/, accessed 2020-06-24.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Wisdom Quotes. 2020. Wisdom Quotes - Get wiser slowly. https://wisdomquotes.com, accessed 2020-11-30.

RiTUAL-UH. 2020. SemEval2020 Task10 Emphasis Selection. https://github.com/RiTUAL-UH/SemEval2020_Task10_Emphasis_Selection, accessed 2020-11-22.

Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1:1–20.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

shallowLearner. 2020. SemEval 2020 - Task 10: Emphasis Selection For Written Text in Visual Media. https://competitions.codalab.org/competitions/20815, accessed 2020-08-15.

Amirreza Shirani, Franck Dernoncourt, Paul Asente, Nedim Lipka, Seokhwan Kim, Jose Echevarria, and Thamar Solorio. 2019. Learning emphasis selection for written text in visual media from crowd-sourced label distributions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1167–1172.

Vipul Singhal, Sahil Dhull, Rishabh Agarwal, and Ashutosh Modi. 2020. Iitk at semeval-2020 task 10: Transformers for emphasis selection. *arXiv preprint arXiv:2007.10820*.

Yu Sun, Shuohuan Wang, Yu-Kun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020a. Ernie 2.0: A continual pre-training framework for language understanding. In *AAAI*, pages 8968–8975.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020b. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.

Chengzhi Zhang. 2008. Automatic keyword extraction from documents using conditional random fields. *Journal of Computational Information Systems*, 4(3):1169–1180.

Kuo Zhang, Hui Xu, Jie Tang, and Juanzi Li. 2006. Keyword extraction using support vector machine. In *international conference on web-age information management*, pages 85–96. Springer.

Named Entity Popularity Determination using Ensemble Learning

Vikram Karthikeyan
Department of ISE
BMS College of Engineering
Bangalore, India
vikram.is17@bmsce.ac.in

Ranjan Samal
Department of Multimodal NLP
Samsung R&D Institute
Bangalore, India
ranjan.samal@samsung.com

B Shrikara Varna
Department of ISE
BMS College of Engineering
Bangalore, India
bshrikara.is18@bmsce.ac.in

Shambhavi B R
Department of ISE
BMS College of Engineering
Bangalore, India
shambhavibr.ise@bmsce.ac.in

Amogha Hegde
DDepartment of ISE
BMS College of Engineering
Bangalore, India
amoghahegde.is17@bmsce.ac.in

Jayarekha P
Department of ISE
BMS College of Engineering
Bangalore, India
jayarekha.ise@bmsce.ac.in

Govind Satwani
Department of ISE
BMS College of Engineering
Bangalore, India
govind.is18@bmsce.ac.in

Abstract

Determining the popularity of a Named Entity after completion of Named Entity Recognition (NER) task finds many applications. The most popular of them being virtual assistants where disambiguating entities without contextual help is crucial. A single named entity could belong to multiple domains, making it necessary for the popularity determination approach to give accurate results. The more accurate results can be used to improve the functioning of virtual assistants. This work studies disambiguation of Named Entities (NE) of Music and Movie domains and resolves popularity considering relevant features like region, movie/music awards, album count, run time etc. Decision Trees and Random Forests approaches are applied on the dataset and the latter ensemble learning algorithm resulted in acceptable accuracy.

1 Introduction

Over the years the use of electronic devices for various tasks and services have become predominant. These days, services such as E-commerce, video and music streaming over the internet are common. Thus, it is crucial that these services provide the users with what they require and also give relevant recommendations. Moreover, search engines should know what results to show based not only on the query string but also consider other factors like demographic, geographic location, user's search history, etc. Identifying entities in a query string is termed as Named Entity Recognition (NER). Due to the vast and diverse collection of data, NER sometimes alone may not be sufficient. In such cases Name Entity Popularity Detection can be used to prioritize results which may be more rel-

Proceedings of the ICON 2020 Workshop on Joint NLP Modelling for Conversational AI, pages 27–32
Patna, India, December 18 - 21, 2020.

evant to the user based on various factors. This is especially used while designing a conversation smart assistant, to provide the user with the best results. For example, if the user asks the virtual assistant to "play wolves" then it has to decide whether to play the song wolves or play the movie wolves without any given context or sentence associated with. In such a scenario information about the user's search history has conventionally been used to resolve the ambiguity and judge what the user really wanted. This process of using data other than the query string to disambiguate between entities with similar names is known as Named Entity Popularity Determination (NEPD). Our objective is to accomplish this task without considering user history.

2 Problem Statement

NEPD aims at determining the popularity of the Named Entities. This approach can be extended for use in Conversation Smart Assistants, helping the application to understand user speech, disambiguate entities and give the best search results. NEPD involves primarily

- Data Mining of features which help to determine the popularity of the target NE.
- Designing an algorithm to predict the popularity from the mined features for various domains. Music and Movie domains are considered in our work.

3 Related Work

Several approaches have been put forward to get better results for NER. Some of these methods involve the use of neural architectures in addition to Bi-LSTM methods. Lample et al. (2016) has developed a method based on transition-based parsing and stack-LSTMs. Building on the approach of ensemble learning for NER, multiple approaches have been combined to obtain a better result (Speck and Ngomo, 2014). The results of various classifier algorithms were integrated (Florian et al., 2003), resulting in significant strides in NER performance.

In the work of (Cucerzan, 2007), information extracted from Wikipedia is stored in two databases, and the entities are mapped together. After entity mappings, a disambiguation component is used for Named Entity Disambiguation (NED). To improve the performance of NED, in addition to Bi-LSTM; other models that use GCN and RNN, along with

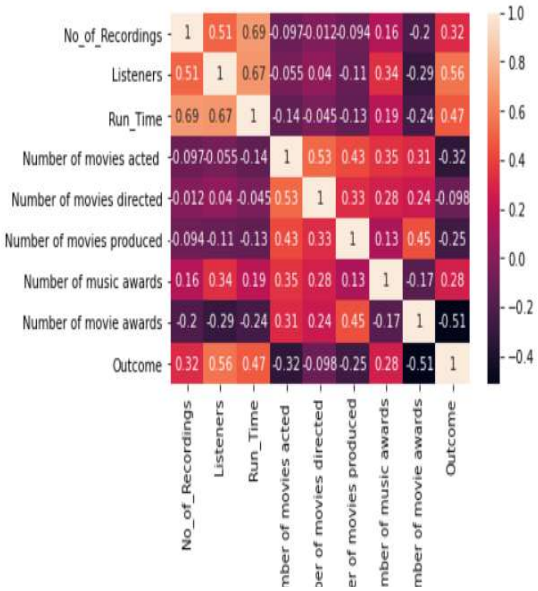

Figure 1: Correlation Matrix of the features

attention have also been experimented upon (Cetoli et al., 2018). Combining graphs and popularity ranking into a single model is another approach towards NED (Alhelbawy and Gaizauskas, 2014) , (Han and Zhao, 2010), where graphs, knowledge bases, are used for Entity Generation, Entity ranking and NED. Joint Embeddings have also been used for improving the results for NED (Yamada et al., 2016).

To enhance the results of the NER models, Entity Popularity models have been proposed (Govani et al., 2013), (Blanco et al., 2013). These models use the personal history of the user, create a ranking of entities, and have resulted in recommendations that are more likely and specifically related to the concerned user. The ranking of entities has been improved by increasing the performance of the Language Model (LM), during Automatic Speech Recognition (Van Gysel et al., 2020). The model used for predicting entity popularity in this paper resulted in improvements of 20% in word error rate.

4 Dataset and Feature Extraction

The next step in dealing with the sparseness was building the correlation and covariation matrix.The correlation matrix, shown in *Figure1* helped decide which features to drop, and which features to retain, to get a better clarity of the data we had collected.

The initial dataset of entities and the domains to which they are to be classified into movie/music

Feature Name	Description	Souce
Type	Genre (applicable to musicians,songs and movies only, and not to actress	IMDb, MusicBrainz
Count of Movies Acted	The number of movies a person has acted in (i a significant role)	IMDb
Count of Movies	Directed Some artists tend to direct the movies they work in.	IMDb
tCount of Movies Produced	Some artists tend to produce the movies they work in. Also includes the number of albums that are self produced by an artist.	IMDb
Release Date	The time description of when a particular song or movie was released. It helps in determining the popularity of that particular work in that decade.	IMDb, MusicBrainz and Last.fm
Region	Place of release of the movie or album or location of concert also refers to the region where an artist is based.	IMDb
Count of Albums	The higher the number, the more weightage for music domain	Last.fm API
cCount of Concerts	The number of live concerts held by the particular artist	Last.fm API
Count of Movie Awards	The number of awards a particular person has in a domain	IMDb
Count of Music Awards	The number of awards a particular person has in a domain	IMDb, MusicBrainz
Run Time	Sum of durations of all the songs released by the person	MusicBrainz

Table 1: Features and their description.

was generated. Considering the domains in the problem statement, a list of applicable features along with the source from which the data for that particular feature can/could be extracted was made. This decision was based on the considerations that a person who is active predominantly in one of the two domains, will have a higher value for the features pertaining to that domain. Additionally, people who are active in both the domains will have values for all the features but some comparatively higher than the other. The list of features is given in *Table 1*.

The problem at hand is finding the popularity of a person in the domain and identification of the domain in which the person is more popular if the person has an existence in both. The idea behind the selection of a feature was based on this problem. The popularity of any person in a domain is based on the works of that person, and the accolades the person has received for their work in the particular field. The feature "Run Time" was included in particular as some of the artists voice for songs in the movies they have acted in. And hence they might have a considerable number of songs under them. This feature will help us to distinguish between a full-fledged singer (music domain) and actors who just give voice for a song in a movie.

The data was collected and a .csv file was formed. Most of the entities in the given dataset were disambiguous, and hence the data collected was sparse as the entity was inclined to one of the domains. To overcome this problem of sparseness, the dataset was first divided into two-parts: one file containing the entities that were the names of the people and/or music bands, while the other contained the entities that were the names of the artwork (movie and/or song tracks). Dividing the single file into two, helped deal with the sparseness a bit as some of the features that were not applicable to that particular category were removed.

There were many challenges in handling the artwork file. Many artworks pertaining to the same

domain had the same name but different artists. A disambiguation had to be done to select the best one of these. Consider the song "Bad Guy". When we tried to collect data regarding it, the result of the search query included 280 songs of the same name in all languages combined. We took the collection of top 10 searches and the data related to each of those songs which shared the common name.

The features "Genre", "Region" and "Release date" were dropped from consideration. The decision to drop them was taken based on the fact that "Genre" didn't work well with entities belonging to the movie domain. The "Release Date" isn't applicable to persons but only to the artwork. Hence these features were considered as not needed and were dropped.

After dropping the not required features from the dataset, the sparseness that still existed was dealt by filling them up with default values which was determined in such a way so as to not change the dynamics of the dataset (i.e., not change the nature of the data).

5 Methodology

Flowchart in *Figure2* shows the systematic approach taken. The data collected from IMDb and MusicBrianz was analyzed. We faced the problem of excessive sparseness after building our dataset as some fields were not applicable for a particular domain and hence had to be left blank. Features like Number of movies acted, Number of movies directed, Number of movies produced would not be applicable for music related entities. Similarly features like release region, release year would not be applicable for movie artists.

We realized adding mean/median values to remove the sparseness would corrupt the dataset as the missing values were not applicable for the entity. For entities like number of movies acted, number of movies directed, number of movies produced we replaced Null values with zero. For the release year entity, we replaced Null values with 2015 and release region with USA to avoid ambiguity as we didn't want these values to influence the classifying decision. We replaced the Null values in the rating columns for movie artists to four to maintain uniformity. We made sure not to replace any missing attribute's value with an extreme value so as to avoid influencing the classifying decision.

We used Label encoding to encode the attribute values as decision trees and random forests do not

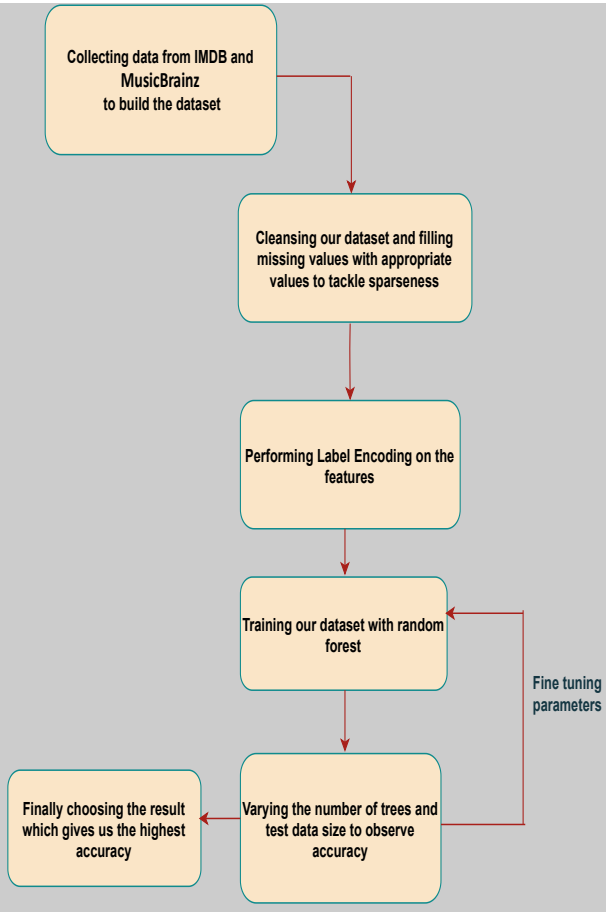

Figure 2: Flowchart of the System Processes

accept string inputs. We used Label Encoding over One Hot encoding because we observed One hot encoding would result in higher data duplication (Multiple Columns). Initially we used decision trees after performing label encoding on the dataset entities. The intuition behind choosing decision trees was that the algorithm generates rules for classification. The input data was not sequential for us to try Machine Learning algorithms like RNN or LSTM. The decision tree was built using the scikit-learn library of python. The algorithm used to build the decision tree was CART, and hence categorical values weren't supported and encoding had to be done. The decision of splitting the node at a level was based on the gini impurity of the features.

To improve the accuracy, we used random forest (Set of decision trees) to reduce overfitting and give better results for new test data. Random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better

than a single decision tree because it reduces the over-fitting by averaging the result and improves the classification accuracy.

6 Experimental Results

On analyzing our dataset, we decided that a machine learning approach would work well for the problem in hand. We initially used decision trees and achieved an accuracy of 86%. *Figure3* represents the decision tree that was built using the CART algorithm. The split of nodes at each level is depicted along with its Gini impurity value.

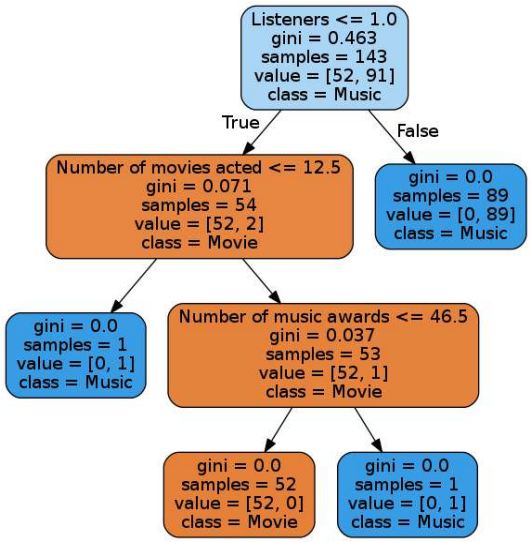

Figure 3: Decision Tree generated

Algorithm	Test Data (%)	Training Data (%)	Accuracy (%)
Decision Tree	20	80	86
Random Forest	20	80	89
Random Forest	30	70	93
Random Forest	40	60	94.3

Table 2: Experimental Results.

To increase the accuracy of classification and prevent overfitting we decided to use random forest. Using the random forest, we obtained an accuracy of 89%. As our dataset was relatively small, increasing the number of trees for random forest did not yield us with better accuracy, so we increased the test data size to have larger data to classify from 20% to 40%. Results are tabulated in Table 2. Finally, we achieved an accuracy of 94.3% percent using random forest with 100 trees.

7 Conclusion

The problem of Named Entity Popularity Determination was to be solved without any contextual data or user history in the domains of music and movies. In virtual assistants, users usually give only named entity without context or a statement associated with it. We built our customized dataset from IMDb and MusicBrainz. The issue of excessive sparseness was solved by filling the missing values in the dataset with generic relevant values and made sure it wouldn't influence the classifying decision. With decision trees, an accuracy of 86% was achieved. To improve the accuracy and prevent overfitting random forests was used. We finally achieved an accuracy of 94.3%. The approach used here to disambiguate ambiguous entities can be extended to other related domains like TV Shows, Podcasts and Radios by collecting relevant features. This approach can be modelled to disambiguate ambiguous entities between various related domains without contextual information.

References

Ayman Alhelbawy and Robert Gaizauskas. 2014. Graph ranking for collective named entity disambiguation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 75–80.

Roi Blanco, Berkant Barla Cambazoglu, Peter Mika, and Nicolas Torzec. 2013. Entity recommendations in web search. In *International Semantic Web Conference*, pages 33–48. Springer.

Alberto Cetoli, Mohammad Akbari, Stefano Bragaglia, Andrew D O'Harney, and Marc Sloan. 2018. Named entity disambiguation using deep learning on graphs. *arXiv preprint arXiv:1810.09164*.

Silviu Cucerzan. 2007. Large-scale named entity disambiguation based on wikipedia data. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 708–716.

Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. 2003. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 168–171.

Tabreez Govani, Hugh Williams, Jamie Buckley, Nitin Agrawal, Andy Lam, and Kenneth A Moss. 2013. Determining entity popularity using search queries. US Patent 8,402,031.

Xianpei Han and Jun Zhao. 2010. Structural semantic relatedness: a knowledge-based method to named entity disambiguation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 50–59.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

René Speck and Axel-Cyrille Ngonga Ngomo. 2014. Ensemble learning for named entity recognition. In *International semantic web conference*, pages 519–534. Springer.

Christophe Van Gysel, Manos Tsagkias, Ernest Pusateri, and Ilya Oparin. 2020. Predicting entity popularity to improve spoken entity recognition by virtual assistants. *arXiv preprint arXiv:2005.12816*.

Ikuya Yamada, Hiroyuki Shindo, Hideaki Takeda, and Yoshiyasu Takefuji. 2016. Joint learning of the embedding of words and entities for named entity disambiguation. *arXiv preprint arXiv:1601.01343*.

Optimized Web-Crawling of Conversational Data from Social Media and Context-Based Filtering

Annapurna P Patil
Department of Computer Science and Engineering
Ramaiah Institute of Technology
Bangalore, India
annapurnap2@msrit.edu

Rajarajeswari S
Department of Computer Science and Engineering
Ramaiah Institute of Technology
Bangalore, India
raji@msrit.edu

Gaurav Karkal
Department of Computer Science and Engineering
Ramaiah Institute of Technology
Bangalore, India
gauravkarkal@gmail.com

Keerthana Purushotham
Department of Computer Science and Engineering
Ramaiah Institute of Technology
Bangalore, India
keerthupuru@gmail.com

Jugal Wadhwa
Department of Computer Science and Engineering
Ramaiah Institute of Technology
Bangalore, India
jugaldeepak@gmail.com

K Dhanush Reddy
Department of Computer Science and Engineering
Ramaiah Institute of Technology
Bangalore, India
dhanushreddy1014@gmail.com

Meer Sawood
Department of Computer Science and Engineering
Ramaiah Institute of Technology
Bangalore, India
sawoodrocket@gmail.com

Abstract

Building Chatbot's requires a large amount of conversational data. In this paper, a web crawler is designed to fetch multi-turn dialogues from websites such as Twitter, YouTube and Reddit in the form of a JavaScript Object Notation (JSON) file. Tools like Twitter Application Programming Interface (API), LXML Library, and JSON library are used to crawl Twitter, YouTube and Reddit to collect conversational chat data. The data obtained in a raw form cannot be used directly as it will have only text metadata such as author or name, time to provide more information on the chat data being scraped. The data collected has to be formatted for a good use case, and the JSON library of python allows us to format the data easily. The scraped dialogues are further filtered based on the context of a search keyword without introducing bias and with flexible strictness of classification.

1 Introduction

Real-world data remains a necessary part of training system models. The digital streams that individuals produce are quite useful in the Data Analysis domain, like natural language processing and machine learning. Social networking applications like Twitter, YouTube and Reddit contain a large volume of data that are quite useful for various algorithms. Naturally, the need to make information easily accessible to all leads to deploying a conversational agent. In order to build a chat model, a huge volume of conversational text data is required.

Twitter is a microblogging service that allows individuals to post short messages called tweets

Proceedings of the ICON 2020 Workshop on Joint NLP Modelling for Conversational AI, pages 33–39
Patna, India, December 18 - 21, 2020.

that appear on timelines. These tweets were limited to 140 characters, which has been later expanded to 280 characters and prone to change again in the future. Tweets consist of two kinds of metadata that are entities and places. Tweet entities are hashtags, user- mentions, images, and places in the real world's geographical locations. Metadata and short prose add to fewer than 280 characters can link to Webpages, Twitter users. Twitter timelines are categorized into the home timeline and user timeline. Timelines are collections of tweets in chronological order. Twitter API uses Representational State Transfer (REST) API to crawl and collect a random set of sample public tweets. The API allows users to explore and search for trending topics, tweets, hashtags, and geographical locations.

YouTube, a video-sharing website, allows users to view, upload, rate, report on videos. It contains a wide variety of videos such as TV show clips, music videos, documentaries. It also provides a platform for users to communicate and describe their thoughts about what they watch through comments.

Reddit is a social news platform where registered users may submit links, images, text, posts and also upvote or downvote the posts posted by other users. Posts are organized based on boards created by the user called subreddit. It is also a platform for web content rating and discussions. Reddit stores all of its content in json format which can be viewed on the browser by extending the reddit link with the extension '.json'.

Real-time datasets are needed to build a model to generate accurate output. As the available datasets are insufficient and do not contain realistic examples, there is a need to build a crawler which would scrape conversational data. Building crawlers for each website would allow collection of conversational data. This would help in the creation of datasets of conversational data.

2 Literature Survey

A Focused crawler is designed to crawl and retrieve specific topics of relevance. The idea of the focused crawler is to selectively look for pages that are relevant while traversing and crawling the least number of irrelevant pages on the web.

Context Focused Crawlers (CFC) use a limited number of links from a single queried document to obtain all relevant pages to the document, and the said obtained documents are relevant concerning the context. This data obtained is then used to train a classifier that would detect the context of documents and allow classification of them into categories based on the link distance from a query to target. Features of a crawler are-

- Politeness
- Speed
- Duplicate Content

Each website comes with the inclusion of a file known as robot.txt. It is a standardized practice where robots or bots are communicated with the website through this protocol. This standard provides the necessary instructions to the crawler about the status of the website and whether it is allowed to scrape the data off of the website. This is used to inform crawlers whether the website can be crawled either partially or fully, if the website cannot be crawled as per the robot.txt then the server blocks any such requests and can even lead to blocking of IP's.

Websites have robot.txt, which prevent the use of crawlers that attempt to scrape large data from their website. Any request for a large amount of data is blocked almost immediately. To prevent such a case where the crawler should not be blocked, a list of publicly available proxy servers as described in Achsan (2014) is used to scale and crawl the website. Twitter is a popular social media platform used for communication. Crawling such a website can be useful to gain conversational data, and such information can be targeted based on the topic; one such example is a perception on the internet of things as shown in Bian J (2017) or Assessing the Adequacy of Gender Identification Terms on Intake Forms as described in Hicks A (2015).

3 Proposed System

3.1 Twitter Crawler

Twitter API provides the tweets encoded in JSON format. The JSON format contains key-value pairs as the attributes along with their values. Twitter handles both the users and as well the tweets as objects. The user object contains attributes including their name, geolocation, followers. The tweet object contains the author, message, id, timestamp, geolocation etc. The JSON file can also contain additional information in the media or links present

in the tweets, including the full Uniform Resource Locator(URL) or link's title or description.

Each tweet object contains various child objects. It contains a User object describing the author of the object, a place object if the tweet is geo-tagged, and entities object, which is an array of URLs, hashtags, etc. Extended tweets include tweets with longer text fields exceeding 140 characters. It also contains a complete list of entities like hashtags, media, links, etc. They are identified by the Boolean truncated field equals true, signifying the extended tweet section to be parsed instead or the regular section of the tweet object.

The retweet object contains the retweet object itself as well as the original tweet object. This is contained in the retweeted status object. Retweets contain no new data or message, and the geolocation and place is always null. A retweet of another retweet will still point to the original tweet.

Quote tweets contain new messages along with retweeting the original tweet. It can also contain a new set of media, links or hashtags. It contains the tweet being quoted in the quoted_status section. It also contains the User object of the person quoting the tweet.

The Twitter REST API method gives access to core Twitter data. This includes update timelines, status data, and user information. The API methods allow interaction with Twitter Search and trends data.

The Twitter streaming API obtains a set of public tweets based upon search phrases, user IDs as well as location. It is equipped to handle GET and POST requests as well. However, there is a limitation on the number of parameters specified by GET to avoid long URLs. Filters used with this API are-

- follow- user IDs of whom to fetch the statuses
- track- specific words to be searched for.
- location- filter tweets based on geo location.

Workflow of Twitter Crawler:

1. Crawling for public tweets

   This project uses Streaming API access to crawl and collect a random sample set of public tweets. These tweets are crawled in real time. These tweets can be filtered with geo location to crawl tweets with respect to a specific region. These sample sets of public tweets are outputted to a json file as seen in Figure 1.


```
{
  "created_at": "Thu Jul 11 14:55:30 +0000
2019",
 "id": 1139331385288495104,
 "id_str": "1139331385288495104",
    "full_text":    "@basura_inutil_    \"For
example, fat ciswomen are less likely to
receive cervical cancer screening... breast
cancer screening... and colorectal cancer
screening than non-far ciswomen.\"\n\nSource
(with citations): https://t.co/gKTPyRZ8s8",
 "truncated": false,
 "display_text_range": [16,231],
 "entities": {
   "hashtags": [],
   "symbols": [],
   "user_mentions": [
     {
       "screen_name": "basura_inutil_",
       "name": "Yung Sough",
       "id": 252837878,
       "id_str": "252837878",
       "indices": [0,15]
     }
   ]
 }
}
```


Figure 1: The above figure depicts a tweet object. It contains all attributes contained within a tweet

2. Crawling for tweets while searching and monitoring a list of keyword

   For searching and monitoring, a list of keywords, search/tweets is used. Streaming API's status/filters are not used as it does not provide previous tweets at all. Search/tweets are used to crawl and provide tweets from at most a week back. This function continues to endlessly crawl for tweets matching the query. Another advantage of using this method is that it does not limit the number of keywords it can track, unlike statuses/filters, which require separate or new instances to search for different keywords.

3. Filtering Tweets with replies

   The crawler filters through the collected public sample of tweets to find specific tweets with replies. It finds the same by checking the 'in_reply_to_status_id' attribute of the collected tweets. It then proceeds to crawl the parent tweet using the 'in_reply_to_status_id'. It outputs the parent and the reply tweet together in the JSON file, as seen in Fig. 2.

4. Filtering Quoted Retweets

   Quoted Retweets or retweets with comments are filtered from the collected set of

public tweets. This is achieved by parsing the collected tweets and searching for 'quoted_status'. The parent tweet is contained within the quoted retweet inside the 'quoted_status' attribute. The crawler outputs the quoted retweet with its contained parent tweet to the JSON file, as seen in Fig. 3.


```
{
 "metadata": {
   "iso_lanaguage_code": "en",
   "result_type": "recent"
 },
 "source": "<a
href=\"http://twitter.com/download/android\"
rel=\"nofollow\">Twitter for Android</a>",
 "in_reply_to_status_id":
1149330296774393863,
 "in_reply_to_status_id_str":
"1149330296774393863",
 "in_reply_to_user_id": 976575708494413824,
 "in_reply_to_user_id_str":
"976575708494413824",
 "in_reply_to_screen_name": "t3rrordactyl",
 "user": {
   "id": 976575708494413824,
   "id_str": "976575708494413824",
   "name": "Aranv Gupta died protesting US
imperialism",
   "screen_name": "t3rrordactyl",
   "location": "Occupied Ohline Territory",
   "description": "Black american bi polyam
cis woman. You're gonna see either
informative political matters or memes here.
Socialist will win. she/her.",
   "url": null,
   "entities" : {
     "description": {
       "urls": []
     }
   }
 }
}
```


Figure 2: In the above diagram, the 'in_reply_to_status' tag used to filter the tweets is highlighted.

3.2 Youtube Crawler

Cascading Style Selector and Python are the primary tools that are used. The crawler works to accept a CSS selector expression as input; the selector compiles to XPath, and many other libraries such as requests.

AJAX interchanges data with the server in the background enabling the web page to be updated. This allows certain sections to be updated as opposed to the entire page. Python is an object-oriented programming language for general-purpose programming. It helps enhance code readability by including the whitespace.

YouTube has server-side rendering with automation,it is impractical to wait for all comments to get loaded in order to extract them. This work uses the fact that YouTube sends an AJAX Request


```
{
 "geo": null,
 "coordinates": null,
 "place": null,
 "contributors": null,
 "is_quote_status": true,
 "quoted_status_id": 1149078505017286661,
 "quoted_status_id_str":
"1149078505017286661",
 "quoted_status": {
   "created_at": "Wed Jul 10 22:10:39 +0000
2019",
   "id": 1149078505017286661,
   "id_str": "1149078505017286661",
   "full_text": "\"Do not let your little
girl ever take this #vaccine. No matter what
happens, don\u2019t let her take this
vaccine\" - @RobertKennedyJr #hpvvaccine
https://t.co/TeeeqRkPE9",
   "truncated": false,
   "display_text_range": [0,141],
   "entities" {
     "hashtags": [
       {
         "text": "vaccine",
         "indices": [44, 52]
       }
     ]
   }
 }
}
```


Figure 3: In the above diagram, the 'quoted_status' tag used to filter the tweets is highlighted.

to a URL in order to get comments. By using that URL, the crawler makes a session with a user agent and sends the AJAX request to the server for a particular video id input given by the user; all the comments are downloaded and stored in a JSON format; each comment and its reptiles are identified by the id it has, replies have the id of the main comment prefixed with its id.

Begin:
  Initialize a new session with user agent
  Get YouTube page with initial comments
  Extract all the comments by sending an AJAX request
  Obtain remaining comments
  Obtain replies
  Allocate the replies cid's prefixed with main comment id
  Parse the json data in the required format
  Store it in a json file.
End

Figure 4: Algorithm Crawl Youtube Comments

3.3 Reddit Crawler

Reddit is a social media platform that serves the purpose of sharing the content of many forms, including multimedia content. A subreddit is defined to cover a topic of interest, for example, sports and users can make posts on sports, and others can comment on these posts.

Usually, scraping Reddit is difficult since Red-

dit easily blocks IP addresses after a certain number of requests. Hence the work done focuses on viewing Reddit pages as a collection of JSON objects that can be simply parsed for the required content type. This overcomes the issue mentioned above of having a single IP address blocked at multiple requests.

Reddit returns a JSON object when the link is extended with '.json', which is passed as an initial request to crawl, and a callback function is called with the response downloaded from the request. In the callback function, the response object is parsed, and the crawler retrieves the comments. Post links are retrieved based on the search term entered and sorted on hot, new, top, rising, or controversial. The number of post links is limited based on the limit factor. Each of these posts contains a set of permalinks containing information about the comments. Each of these permalinks is parsed, are traversed, and a JSON object is retrieved. Obtained JSON objects are parsed to retrieve the comments.

Scrapy, a web scraping framework, is used to extract structured content from the Reddit page. Libraries such as JSON, time are used. The JSON library provided by python is used to parse JSON objects returned from Reddit links. The crawler output can be seen as in Fig. 5.


```
  {
   "id": "exrxaiu",
   "user-name": "K-369",
   "text": "They should open a category for
performance enhanced athletes only. Only ban
them for serious narcotics but allow hormone
enhancers and steroids.\nThen we'll see the
human body seriously pushed to the limit",
   "time-stamp": "Fri Aug 23 07:43:58 2019",
   "likes": 2,
   "original":
"https://www.reddit.com/r/sports/comments/ct
yn6o/the_worlds_fastest_man_may_be_banned_fr
om_the/exrxaju/.json",
   "topic": "science",
   "depth": 1,
   "replies": [
     {
       "id": "exsgaef",
       "user-name": "VilleKivinen",
       "text": "That category already exists,
it's called the Olympics.",
       "time-stamp": "Fri Aug 23 15:11:43
2019",
       "likes": 1,
       "original":
"https://www.reddit.com/r/sports/comments/ct
yn6o/the_worlds_fastest_man_may_be_banned_fr
om_the/exrxaju/.json",
       "topic": "science",
       "depth": 0,
       "replies": []
     }
   ]
  }
```


Figure 5: Crawler output in JSON format

4 Context Based Dialogue Filtering

The tool used in the comment filtering module is python libraries centered around handling and manipulating the data obtained from the crawlers. The filtering model uses the pandas' library to read the raw data from a .csv form of the output json data from the crawlers. The filtering model uses the nltk library to clean our data, tokenize the data, and remove stop words.

The outputs collected from the crawlers are all initially in JSON files. These are then converted into a uniformly structured csv type file. The main technology used is in the form of the bag of words model used to analyze the importance of each generated token within the context of the extracted data.

In the implementation, note that input is the csv file, and output is a cleaned and appended list of comments and replies. The comments and replies are first to read into a data frame following which the following cleaning methods are applied: Convert to lowercase "[/()\[\]\—@,;" symbols are replaced by a space "0-9a-z +_" symbols are removed

Stopwords are removed according to the 'English' stopwords from the nltk stopwords library

The first 30 comments are then analyzed to generate a list of tokens, and their frequencies are counted. Tokens with "" are given high preference, and a high-frequency list of words is taken as a subset of the original list. Then all the comments and replies from the data frame are cross-referenced. Entries that do not contain any of the words in the list of high-frequency words are rejected. The remaining entries which have been filtered are the output.

Begin
Convert .json files to a standard .csv file.
Extract the comments and replies columns into a dataframe.
Clean the raw data.
Tokenize the entries in the dataframe and calculate the word frequency for each unique word.
Generate a high frequency word list as a subset of the former.
Cross-refer the remaining entries and filter in those that include a token from the high frequency word list.
This output is cleaned, filtered data.
End

Figure 6: Algorithm Comment filtering

5 Analysis and Comparison

5.1 Twitter

Standard twitter crawlers use the Streaming API statues/filters, which do not provide old tweets. The other caveat is that it can only track a limited number of keywords. So, if the user has a lot to track, he will need to have many separate instances, each tracking different parts of the keywords.

The proposed Twitter crawler is beneficial in that it uses search/tweets to get old tweets. It searches for a portion of the keyword list at a time. The Streaming API is used to collect a random sample of real-time public teams. The REST API is used to filter and obtain tweets with respect to specified filters. The implemented algorithm filters the tweets to obtain tweets relevant to conversational data.

5.2 YouTube

Selenium, a vast tool for scraping, can crawl YouTube. YouTube has server-side rendering, which loads the website first upon which it can be scraped for data. This approach's problem is that the website has to be loaded upon which JavaScript has to be used to load more comments on the web page.

The novelty in this approach lies in that by analyzing how YouTube retrieved its comments, and the same approach can be used by sending AJAX requests to the URL https://youtube.com/id/comments. Since sections of the page that does not include comments are not downloaded, the crawler saves time and resources over the downloaded contents.

5.3 Reddit

Reddit being a dynamic website and communicates through rest API; selenium is a powerful web scraping tool that can be used to scrape dynamic websites. While simulating the browser, crawlers can mimic the scrolling of the pages, clicking to read more comments. Since the speed of retrieval will depend on the speed at which each section of the page loads, it is inefficient as the next load will only start after the previous load and the cursor scrolls further down.

Scrapy is faster and robust in handling errors but does not allow the crawling of dynamic sites. Scrapy can be provided with a Reddit URL extended by the '.json' link and get all internal links consisting of comments and replies. Instead of traversing the Reddit page using html and selectors, which would be time-consuming, a simple scraper has been built using the functionality of scrapy.

5.4 Filtering of extracted data

The filtering model is a simple algorithm that avoids excessive computation as it does not look outside the extracted data for filtering conditions. It works more along the lines of "sticking to the topic" by using the high-frequency terms and hashtags from the extracted data itself. A sample of filtered and unfiltered data for a Twitter search term "cancer" is attached below. One of the two variables to note here is the number of comments used to generate the high-frequency list - . Ideally, it should depend on the volume of extracted data, at least 2-5% of the total volume.

The other variable is the frequency number-used to classify a token as highly frequent. This would depend on the number of tokens used to create a high-frequency list. Approximately the first 80% of the tokens, when arranged in ascending order in terms of their occurrence frequency, need to be rejected. Samples of such filtering are seen in Figure 7 and Figure 8.

```
The extracted comments are:
  "looking another reason put cigarettes
according american cancer society smoking
increases risk cervical cancer also reduce
risk getting hpv vaccine scheduling regular
pap smear #papsmear #cervicalcancer
#savannahga"
  "circumcision greatest scam fell could
choose age knowledge would say"
  "give freaking break"
  "misleading claim take look list services
offered even ones sound like something
family doctor might arent ie pediatric care
doesnt include vaccinations prevention
doesnt include routine cancer screening like
pap tests 2"
```

Figure 7: Accepted and cleaned comments. This is a sample of accepted comments that were cleaned prior filtering for the searched keyword, "cancer".

Thus we see that this filtering algorithm achieves highly flexible strictness with respect to which comments are accepted and with no bias. Since this is a simple implementation of a modified bag of words model, it is computationally light in comparison with ML models that do the same.

6 Conclusion

The proposed crawler can fetch multi-turn dialogues from Twitter, YouTube. The crawler can scrape conversational data from Twitter, YouTube.

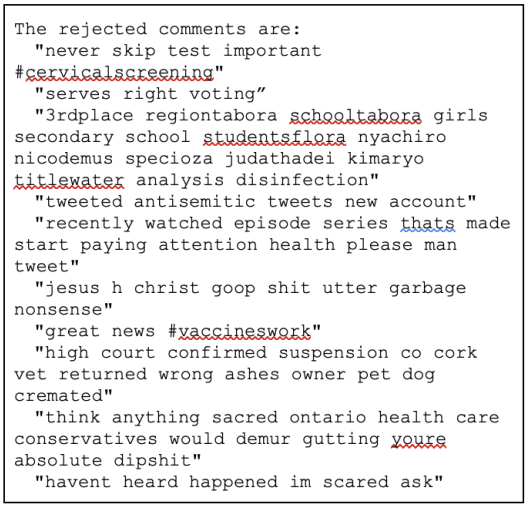

Figure 8: Cleaned but rejected comments. This is a sample of rejected comments that were cleaned before filtering for the searched keyword, "cancer".

Unstructured data retrieved from the crawler is converted to well-formatted data. Streaming API has been used to crawl Twitter and retrieve random sets of public tweets. Quoted Retweets or retweets with comments are filtered from the collected set of public tweets. The quoted retweet, with its contained parent tweet, is outputted to the JSON file. YouTube crawling is made easy without any limitations as like which the YouTube data v3 API has, and getting comments has never been this easy and fast before. Reddit crawler parses the JSON object from the response downloaded from the spider's request and can get the comments of the posts.

References

Wahyu Achsan, Harry Wibowo. 2014. A fast distributed focused- web crawling. *Procedia Engineering*, pages 492–499.

Sandip Chauhan Ayar Pranav. 2015. Efficient focused web crawling approach for search engine. *International Journal of Computer Science and Mobile Computing*, 4:545 – 551.

Hicks A Yuan J He Z Xie M Guo Y Prosperi M Salluom R Modave F Bian J, Yoshigoe K. 2016. Mining twitter to assess the public perception of the "internet of things". *PLoS One*, 11(7).

Salloum RG Guo Y Wang M Prosperi M Zhang H Du X Ramirez-Diaz LJ He Z Sun Y Bian J, Zhao Y. 2017. Using social media data to understand the impact of promotional information on laypeople's discussions: A case study of lynch syndrome. *J Med Internet Res*.

Satish Kumar Dhiraj Khurana. 2012. Web crawler: A review. *International Journal of Computer Science Management Studies*, 12(1).

Rutherford M Malin B Xie M Fellbaum C Yin Z Fabbri D Hanna J Bian J Hicks A, Hogan WR. 2015. Mining twitter as a first step toward assessing the adequacy of gender identification terms on intake forms. *AMIA AnnuSymp Proc*.

Vassiliki Angelis LefterisVakali Athena K. Paparrizos, IoannisKoutsonikola. 2010. Automatic extraction of structure, content and usage data statistics of websites. pages 301–302.

M. Singhal M. Bahrami and Z. Zhuang. 2015. A cloud-based web crawler architecture. *International Conference on Intelligence in Next Generation Networks)*, pages 216–223.

Varnica. Stockmeyer Mini Singh Ahuja, Dr. Jatinder Singh Bal. 2014. Web crawler: Extracting the web data. *International Journal of Computer Trends and Technology (IJCTT)*, 13(3):132–137.

Károly Nemeslaki, AndrásPocsarovszky. 2011. Web crawler research methodology.

Christopher Olston and Marc NajorkInfolab. 2010. Web crawling. *Stanford university, Foundations and TrendsR in Information Retrieval*, 4(3):175–246.

Felix K Akorli Pavalam S M, S V Kashmir Raja and Jawahar M. 2011. A survey of web crawler algorithms. *IJCSI International Journal of Computer Science Issues*, 8(1).

Jawahar M. Pavalam S. M., S. V. Kasmir Raja and Felix K. Akorli. 2012. Web crawler: Extracting the web data. *IJMLC*, 2(4):531–534.

Hui Shen Xiaoyu Li Wenjie Cao ZiqiangNiu Shuzi. Yanran, Li Su. 2017. Dailydialog: A manually labelled multi-turn dialogue dataset.

A character representation enhanced on-device Intent Classification

Sudeep Deepak Shivnikar, Himanshu Arora, Harichandana B S S
Samsung R & D Institute, Bangalore
{s.shivnikar, him.arora, hari.ss}@samsung.com

Abstract

Intent classification is an important task in natural language understanding systems. Existing approaches have achieved perfect scores on the benchmark datasets. However they are not suitable for deployment on low-resource devices like mobiles, tablets, etc. due to their massive model size. Therefore, in this paper, we present a novel light-weight architecture for intent classification that can run efficiently on a device. We use character features to enrich the word representation. Our experiments prove that our proposed model outperforms existing approaches and achieves state-of-the-art results on benchmark datasets. We also report that our model has tiny memory footprint of ~5 MB and low inference time of ~2 milliseconds, which proves its efficiency in a resource-constrained environment.

1 Introduction

In a time where consumers and businesses alike are constantly adopting new technologies in hope of increasing efficiency and convenience, the intelligent virtual assistant (IVA) has been an immediate success[1]. For a high-quality IVA, it is very crucial to understand the intentions behind customer queries, emails, chat conversations, and more in order to automate processes and get insights from customer interactions. Thus research interest in Intent detection is on the rise.

Intent classification is an important task in Natural Language Understanding (NLU) systems which is the task of assigning a categorical intent label to an input utterance. Most IVAs use cloud-based solutions and mainly focus on accuracy rather than model size. But due to factors like privacy and personalization, there is a need for deploying models on device and thus on-device intent classification is significant. The current state-of-the-art models are highly accurate on benchmark datasets. However, most of these models have a huge number of parameters and use complex operations. Due to these reasons, they are not suitable for deployment on low-resource devices like mobiles, tablets, etc.

Gartner[2] predicts that by 2020, 80% of the smartphones shipped will have on-device AI capabilities. For this, there is a need for light-weight, fast and accurate models that can run efficiently in a resource-constrained environment. Thus, in this paper, we propose an on-device intent classification model.

In our proposed model, we use character features along with word embeddings to get enriched word representations. We use Long Short Term Memory Recurrent Neural Network (LSTM-RNN) (Hochreiter and Schmidhuber, 1997) to obtain the context vector for the input utterance. Further, we benchmark our model against publicly available ATIS and SNIPS datasets. Our experiments show that the use of character features has resulted in improved accuracy on the benchmark datasets.

The major contributions of this paper are given below.

- We propose a novel on-device architecture for Intent Classification which uses character features along with word embeddings.
- We benchmark our model against publicly available ATIS and SNIPS

[1] https://www.statista.com/topics/5572/virtual-assistants/

[2] https://www.gartner.com/en/newsroom/press-releases/2018-03-20-gartner-highlights-10-uses-for-ai-powered-smartphones

Proceedings of the ICON 2020 Workshop on Joint NLP Modelling for Conversational AI, pages 40–46
Patna, India, December 18 - 21, 2020.

datasets and achieve state-of-the-art results.

- We measure the system-specific metrics like RAM usage and inference time and show that our proposed model is efficient for low-resource devices.

The rest of the paper is organized as follows. In section 2, we give a brief overview of existing approaches for the task. We describe our approach in detail in section 3. Experimental results are presented in section 4. In section 5, we conclude and discuss future work.

2 Related work

Intent classification is the task of predicting an intent label for the given input text. It is a well-researched task. Early research include maximum entropy Markov models (MEMM) by Toutanova and Manning (2000). Haffner et al. (2003) and Sarikaya et al. (2011) have approached this task using Support Vector Machines (SVM).

Another popular model used were CRF based methods. Lafferty et al. (2001) first proposed Conditional (CRF) to build probabilistic models for segmentation and labelling sequence data which was proved to perform better over MEMMs. Following this, Triangular-chain conditional random fields was proposed by Jeong and Lee (2008) which is used to jointly represent the sequence and meta-sequence labels in a single graphical structure. This method outperformed the base model.

Purohit et al. (2015) have demonstrated the effectiveness of using knowledge-guided patterns in short-text intent classification. Sridhar et al (2019) have proposed the use of semantic hashing for intent classification for small datasets.

Recently joint models for intent classification and slot filling have been developed. Taking inspiration from TriCRF, Xu and Sarikaya (2013) proposed a CNN-based TriCRF for joint Intent and Slot filling. This was a neural network version of TriCRF which outperformed the base model by 1% for both intent and slot. A joint model using Gated Recurrent Unit (GRU) and max pooling for intent detection and slot filling was developed by Zhang and Wang (2016). Following this, Hakkani-Tur et al. (2016) and Liu and Lane (2016) also developed a joint model using recurrent neural networks. To model the relationship between the intent and slots, Goo et al. (2018) and Li et al. (2018) have used gate mechanism. Wang et al. (2018) have proposed Bi-model based RNN semantic frame parsing network structures by considering the cross-impact of both the tasks. Zhang et al. (2019) have used capsule networks that considers the hierarchical relationships between words, slots, and intents. E et al. (2019) have used SF-ID network to provide bidirectional interrelated mechanism for intent detection and slot filling tasks. Qin et al (2019) have used stack-propagation framework to better model the relationship between slots and intents. They further use BERT with their approach to achieve the current state-of-the-art results.

Although above mentioned joint models achieve impressive results on benchmark datasets, they are inefficient for the applications where only intent information is sufficient. Also, their heavy architecture and large model size make their on-device deployment difficult. Most of these models use multiple layers of operations that result in higher RAM usage and inference time. Our proposed model is light-weight, fast, and accurate, which makes it highly efficient for deployment on low-resource devices.

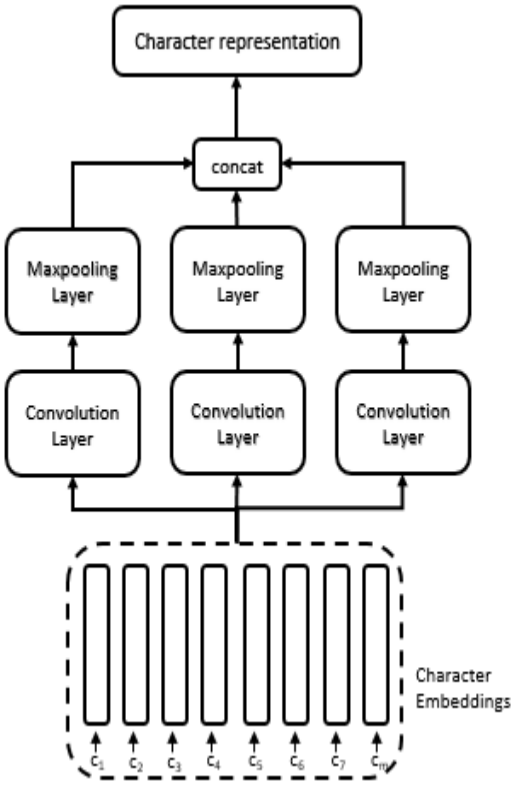

Figure 1: Architecture of character feature extractor

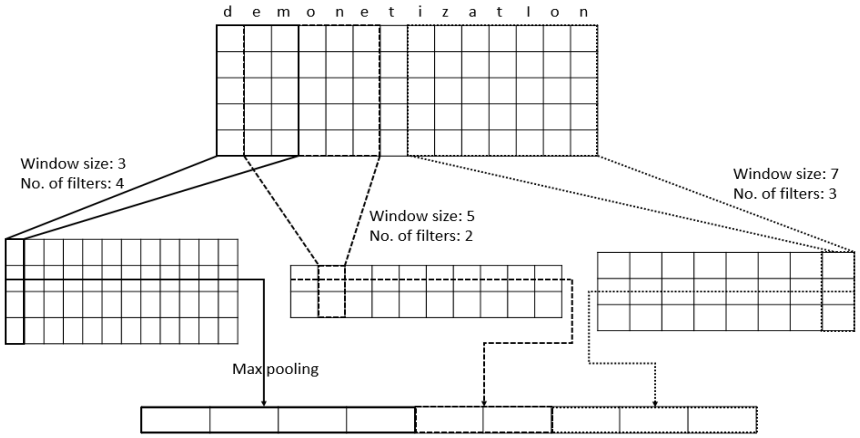

Figure 2: Illustration of character feature extraction.

3 Approach

In this section, we discuss our approach in detail. We use a light-weight architecture that can be efficiently deployed on device. We use character-level representation along with word embedding to enrich word-level representation.

Character representation: The use of character-level features to represent a word has proven useful for multiple NLP tasks. It has been used for language modeling (Kim et al., 2015), parts of speech (POS) tagging (Santos and Zadrozny, 2014), named entity recognition (NER) (Santos and Guimarães, 2015), etc. The use of character level features makes the model robust towards spelling mistakes. Since representations are formed using characters, out-of-vocabulary (OOV) words also get representation which can be further fine-tuned. It also helps to get similar representations for words with common root/prefix. For example consider the following three words: petrify, petrifies, and petrifying. These three words get similar representation using character features as they share a common prefix 'petrif'.

The architecture used to get character representation is depicted in Figure 1. Each word is a sequence of characters. Character embeddings are used to encode this information. Character embeddings are initialized randomly and learned during training. These character embeddings are fed to 3 convolution layers. Convolution layers have different convolution windows which help them to capture different character features. Max-pooling is performed on the output of convolution to select dominant features. The output of max-pooling layers is concatenated to get the character level representation.

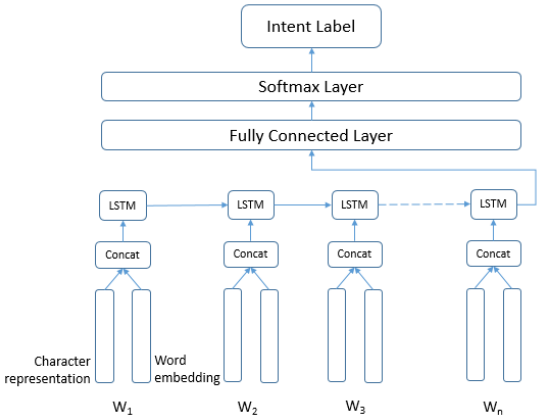

Figure 3: Architecture of proposed model

An illustration of the above-mentioned process is shown using figure 2. For illustration, we use character embedding size of 5. Convolution window sizes and filter sizes are set to (3, 5, 7) and (4, 2, 3) respectively. Max-pooled vectors are concatenated to get character features.

Word level representation: We use word embeddings to capture semantic information. They are initialized with pre-trained embeddings and fine-tuned during training. Pre-trained word embeddings are trained on a huge corpus and hence capture the semantic representation of the word well, which can be further fine-tuned. The use of pre-trained word embeddings helps the model to converge quickly resulting in lower training time. We use pre-trained glove embeddings (Pennington

Intent	Example
wish	Wish you a very happy birthday James!
invitation	You are invited to our wedding. Please attend.
announcement	We are going to be parents!
love	Love you to the moon and back dear!
thank	Thank you for your unconditional love and useful advice!
miss	I hope to see you pretty soon as I miss you way too much, dear.
sorry	I hope you can accept my apology and get rid of my guilt. Sorry
job posting	We are hiring! A Registered Pharmacist, is needed at our Pune office
sale	Get 39% off on Super Skinny Women Blue Jeans. Hurry up Stock is limited
quotes	The more you fall, the more stronger you become for getting up. Never give up no matter what.

Table 2: Details of custom datasets.

et al., 2014). Final word-level representation is obtained by concatenating word embedding with the character-level representation of the word. These concatenated embeddings are fed as input to the encoder.

An encoder is used to get the semantic vector representation for a given input utterance. We use Long Short Term Memory Recurrent Neural Network (LSTM-RNN) as an encoder. LSTM reads the inputs in the forward direction and accumulates rich semantic information. As a complete sentence passes through LSTM, its hidden layer stores the representation for the entire input sentence (Palangi et al., 2015). This sentence representation is used classification.

We use a fully connected layer followed by a softmax layer for classification. The fully connected layer learns function from sentence representation fed to it by the LSTM layer. Softmax layer gives the output probabilities for intent labels. We have illustrated this architecture using figure 3.

4 Experimental Results

In this section, we share the details of the benchmark and custom datasets, describe the training set-up, present experimental results, and compare our model with existing baselines.

Attributes	ATIS	SNIPS
No. of intents	21	7
Vocabulary Size	722	11241
Train set size	4478	13084
Test set size	893	700
Validation set size	500	700

Table 1: Details of benchmark datasets.

4.1 Datasets

To compare our model with existing approaches, we benchmark it against two public data sets. First is the widely used ATIS dataset (Hemphill et al., 1990), which contains audio recordings of flight reservations. The second dataset is the custom-intent-engines dataset called SNIPS (Coucke et al., 2018) which is collected by Snips voice assistant. Details about both the datasets can be found in table 1. SNIPS dataset is more complex as compared to ATIS dataset because of multi-domain intents and relatively large vocabulary. We use the datasets that are pre-processed by Goo et al. (2018) with the same partition for train, test, and validation set.

Custom dataset: We have also curated a custom dataset. For initial data creation, we use user trial and scrape webpages. We define 10 intent

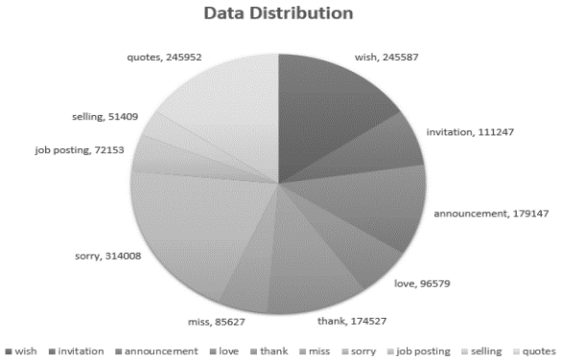

Figure 4: Data distribution in custom dataset

labels to annotate custom dataset. These intent labels and their examples are given in table 2. However, the size of the data that is collected is not sufficient to train a neural network. Therefore we use different data augmentation techniques to increase data size. We use transformer-based data augmentation followed by synonyms replacement. In transformer-based data augmentation, we fine-tune a pre-trained BERT (Devlin et al., 2018) model on collected data. We use this fine-tuned BERT for making predictions on a huge unlabeled corpus. We only consider sentences that are classified with high confidence. We verify these sentences to ensure that predictions by BERT are correct. Further, we use synonyms replacement for augmentation. In a sentence, we randomly replace 30% of the words with their synonyms. The final data distribution is given in figure 4. For testing, we have curated a set of 100 sentences manually.

4.2 Training

We use the same set of parameters for training model on both benchmark datasets. We fix maximum sequence length to 25. We initialize word embedding with 50 dimensional pre-trained GloVe embedding. Character embedding size is set to 15. Kernel sizes are set to 3, 4, and 5 and filter sizes are set to 10, 20, and 30 in 3 convolution layers. LSTM layer has 128 units. Categorical cross-entropy is used for loss computation & Adam optimizer (Kingma and Ba, 2014) is used to minimize loss. The batch size is set to 16. Constant learning rate of 0.001 is used. Models are trained for 10 epochs.

Following Goo et al. (2018) we use accuracy as the metric for the evaluation. After each epoch, we evaluate the performance of the model on validation set. The model performing best on validation set is then evaluated on test set. To address the issue of random initialization, we repeat this process 20 times and consider the average accuracy for analysis.

For custom dataset, we use only 50K sentences per intent label. We also limit word vocabulary size to 12K most frequent words. Batch size is set to 64. Rest all parameters and hyper-parameters remains same as training benchmark datasets.

4.3 On-device deployment

We use TensorFlow (Abadi et al., 2016) to build all our models. We use Tensorflow Lite (tflite) to support on-device execution. The trained models on SNIPS and ATIS datasets are converted to tflite format. The size of the models is reduced by post-training quantization. The final size of our models trained on ATIS and SNIPS is 172 KB and 686 KB respectively. The size of the model trained on custom data set is 786 KB.

Model	ATIS	SNIPS
Joint Seq (Hakkani-Tur et al., 2016)	92.6	96.9
Attention BiRNN (Liu and Lane, 2016)	91.1	96.7
Slot-Gated Full Atten (Goo et al., 2018)	93.6	97.0
Slot-Gated Intent Atten (Goo et al., 2018)	94.1	96.8
Self-Attentive Model (Li et al., 2018)	96.8	97.5
Bi-Model (Wang et al., 2018)	96.4	97.2
CAPSULE-NLU (Zhang et al., 2019)	95.0	97.3
SF-ID Network (E et al., 2019)	96.6	97.0
Stack-Propagation (Libo et al., 2019)	96.9	98.0
Stack-Propagation + BERT (Qin et al., 2019)	97.5	99
Our Model	99.53	98.95

Table 3: Performance of our model compared to current existing approaches

4.4 Baselines

We compare our model with existing deep learning based models for intent classification. They are as follows: Joint Seq (Hakkani-Tur et al., 2016), Attention BiRNN (Liu and Lane, 2016), Slot-Gated Attention (Goo et al., 2018), Self-Attentive Model (Li et al., 2018), Bi-Model (Wang et al.,

Metrics	ATIS	SNIPS
Model Size	172 KB	686 KB
Inference Time	1.87 ms	1.9 ms
RAM	4850 KB	4822 KB

Table 4: On-device model performance

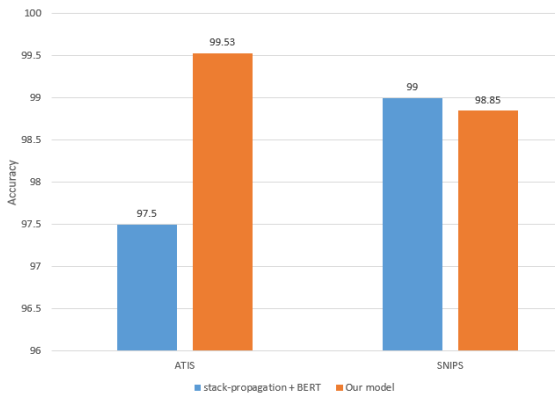

Figure 5: Accuracy comparison of our model with current state of the art

2018), Capsule-NLU (Zhang et al., 2019), SF-ID network (E et al., 2019) and Stack-Propagation (Qin et al., 2019).

For all the baselines, we utilize the results reported by Qin et al. (2019).

4.5 Results

Detailed performance comparison of our model with existing approaches is shown in table 3. Our model achieves an average accuracy of 98.95% and 99.53% on SNIPS and ATIS datasets respectively. The variance of 0.026 on SNIPS and 0.01 on ATIS datasets prove the robustness of our model. Our model outperforms the current state-of-the-art Stack Propagation framework + BERT by 2.03% on ATIS dataset. On SNIPS dataset, our model achieves results comparable to the state-of-the-art and outperforms all other approaches. It is worth noticing that our model has much fewer parameters as compared to the state-of-the-art model. The model trained on custom dataset achieves 98% accuracy on custom test set.

We also measure the system-centric metrics for our models which are presented in table 4. We use a Samsung galaxy A51 device (4 GB RAM, 128 GB ROM, Android 11, Exynos 9611) for these experiments. Inference time includes the time required to pre-process the input text, tokenization, model execution, and label determination. We infer complete test set of datasets on device and report its average inference time. As stated in table 4, our models have an inference time of ~2 milliseconds. We also report maximum RAM usage during on-device inferencing is less than 5 MB.

All the above-mentioned results prove that our model is not only accurate but it also has low inference time and RAM usage. This proves that our model is efficient for running on low-resource devices.

5 Conclusion

In this paper, we present an on-device intent classification architecture that uses character level features to enrich the word representation. Our experiments prove the effectiveness of our model as it achieves state-of-the-art results on benchmark datasets. System centric metrics like RAM usage and inference time shows that our model is fast and light-weight to be deployed on low-resource devices. For future work, we want to extend this approach for slot filling and experiment with a joint model for on-device intent detection and slot filling.

References

Alice Coucke, Alaa Saade, Adrien Ball, Theodore ´ Bluche, Alexandre Caulier, David Leroy, Clement ´ Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190.*

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454.*

Changliang Li, Liang Li, and Ji Qi. 2018. A self-attentive model with gate mechanism for spoken language understanding. In *Proceedings of EMNLP.*

Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990.*

Chenwei Zhang, Yaliang Li, Nan Du, Wei Fan, and Philip Yu. 2019. Joint slot filling and intent detection via capsule neural networks. In *Proceedings of ACL.*

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and YunNung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of NAACL.*

Cicero Dos Santos, Bianca Zadrozny. 2014. Learning Character-level Representations for Part-of-Speech Tagging. In *Proceedings of the 31st International Conference on Machine Learning, PMLR 32(2):1818-1826, 2014.*

Cicero Nogueira dos Santos, Victor Guimarães. 2015. Boosting Named Entity Recognition with Neural Character Embeddings. *arXiv preprint arXiv:1505.05008, 2015.*

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Dilek Hakkani-Tur, Gokhan Tur, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and YeYi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. In *Interspeech*, pages 715–719.

Haihong E, Peiqing Niu, Zhongfu Chen, and Meina Song. 2019. A novel bi-directional interrelated model for joint intent detection and slot filling. In *Proceedigs of ACL*.

Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, Rabab Ward. 2015. Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval. *arXiv preprint arXiv:1502.06922*

Hemant Purohit, Guozhu. Dong, Valerie Shalin, Krishnaprasad Thirunarayan, and Amit Sheth. 2015. Intent classification of short-text on social media, In *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on*. IEEE, 2015, pp. 222–228.

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

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

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML'01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.*

Kristina Toutanova and Christopher D. Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pages 63–70, Hong Kong, China. Association for Computational Linguistics*

Kumar Shridhar, Ayushman Dash, Amit Sahu, Gustav Grund Pihlgren, Pedro Alonso,Vinaychandran Pondenkandath, György Kovacs, Foteini Simistira, and Marcus Liwicki. 2019. Subword semantic hashing for intent classification on small datasets. In *Proceedings of International Joint Conference on Neural Networks (IJCNN), Jul. 2019.* pp. 1–6.

Libo Qin, Wanxiang Che, Yangming Li, Haoyang Wen, Ting Liu. 2019. A Stack-Propagation Framework with Token-Level Intent Detection for Spoken Language Understanding. In *EMNLP, 2019*

Martin Abadi et al. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. CoRR, vol. abs/1603.0, 2016, [Online]. Available: *http://arxiv.org/abs/1603.04467*.

Minwoo Jeong and G.G. Lee. 2008. Triangular-chain conditional random fields, *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, pp. 1287– 1302, September 2008.

Patrick Haffner, Gokhan Tur, and Jerry H Wright. 2003. Optimizing SVMs for complex call classification. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings (ICASSP'03). 2003 IEEE International Conference on*. IEEE, volume 1, pp. I-632.

Puyang Xu, and Ruhi Sarikaya. 2013. Convolutional neural network based triangular crf for joint intent detection and slot filling. *IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013

Ruhi Sarikaya, Geoffrey E Hinton, and Bhuvana Ramabhadran. 2011. Deep belief nets for natural language call-routing. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, pages 5680–5683.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural computation*. 9(8):1735–1780

Xiaodong Zhang and Houfeng Wang. 2016. A joint model of intent determination and slot filling for spoken language understanding. In *Proceedings of IJCAI.*

Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. 2015. Character-Aware Neural Language Models. *arXiv preprint arXiv:1508.06615, 2015.*

Yu Wang, Yilin Shen, and Hongxia Jin. 2018. A bimodel based rnn semantic frame parsing model for intent detection and slot filling. In *Proceedings of ACL.*

Author Index