Multi-Task Deep Learning for Affective Content Detection from Text

by

Weizhao Xin

Thesis submitted to the
Faculty of Engineering
In partial fulfillment of the requirements
For the MCS degree in
Master of Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Deep learning (DL) is a subset of machine learning and artificial intelligence. It has broad adaptability to most types of tasks, including but not limited to text classification and identification of objects in images and videos. It can generate more powerful models compared with legacy machine learning methods. Multi-task Learning (MTL) is an approach that improves generalization by using the domain information which is contained in the training signals of related tasks as an inductive bias. When we apply Multi-task Learning to Deep Learning, the method is called Multi-task Deep Learning.

We focus on deep learning for natural language processing, in particular on how multi-task learning can be used to improve the performance on several tasks at the same time. We present two experiments that deploy Multi-task Deep Learning for detecting affect information from texts.

In experiment 1, we propose a hard parameter sharing multi-task deep learning model for the task of detecting happiness ingredients. For training Deep Learning classifiers, the two primary classes, "agency" and "social", meaning whether the author is in control or the moment involves other people, are treated as two separate tasks while "concept", meaning the categories of the moment, is serverd as an auxiliary task. Then, we train a multi-task deep learning classifier to see if the shared knowledge among the three tasks can be used to improve the overall results. In addition, we compare several models that use different kinds of word embeddings: different dimensions of the vectors, fixed versus trainable embeddings, initialized randomly or with pre-trained embeddings.

In experiment 2, we compare several different multi-task deep learning models on the task of six labels: *Information_disclosure, Emotional_disclosure, Support, General_support, Info_support*, and *Emo_support*. The labels mean that the texts contain informational or emotional disclosure of a person, or express informational or emotional supprtiveness, which can also be catchphrases. We propose a novel way to employ the multi-task deep learning model for the task of detecting disclosure and support, called Venn-diagram-based fragment MTL model. We calculate all possible logical relations between the six labels, represented in a Venn diagram. Based on it, the six labels are distributed to multiple fragment layers. Then, a multi-task deep neural network is built on these layers.

We showed that our multi-task learning model has a stronger ability to represent multi-label tasks over multiple single-task learning models, and using pre-trained trainable embeddings with auxiliary tasks can get the best results. Furthermore, among different multi-task deep learning structures, our model based on Venn diagrams achieved better performance than regular multi-task deep learning and obtained the best results in the CL-Aff Shared Task 2020 for the disclosure labels.

Acknowledgements

First, I wish to express my deepest gratitude to my supervisor, Professor Diana Inkpen, for the continuous support of my Master's study and related research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisor for my Master's study.

I thank my fellow labmates for the help and direction in experiments and thesis writing, for the stimulating discussions, and for all the fun we have had in the last two years. In particular, I am grateful to Prasadith Buddhitha for enlightening me on the first glance of research and Qianjia Huang to help me acquaint myself with the laboratory.

I wish to extend my special thanks to the support staff in the Faculty of Engineering of the University of Ottawa, for providing me with all the necessary facilities for the research.

Last but not least, I would like to thank my parents for supporting me spiritually throughout writing this thesis and my life in general.

Dedication

I dedicate this to my mother and father.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Social media is the product of the information technology revolution and greatly changes the world. Nowadays in our society, the use of social media has become a major daily activity of humans. People use it for access to the news and information in the world and social interaction with others. It uses texts, graphics and videos to connect us to the outside, locally or globally.

Among all possible mediums in social media, text may be the most popular one that been used every day. We use messages to contact our families, share the moments on twitter, read the news on the websites, etc. Hundreds of millions of texts are generated every day. Most of the texts are valueless and discarded, while some of them contain useful information. However, no one in the world has the ability to deal with all of them. The imbalance between the speed of information generating and the inability to process them promotes the development of a hot subject, natural language processing.

Natural language Processing (NLP) is a branch of artificial intelligence that enables computers to read, understand, interpret and manipulate human language. NLP attracts interest from numerous disciplines besides computer science, in its pursuit to fill the gap between human communication and computer understanding. Many natural language processing techniques depend on machine learning (ML) to derive significance from human languages by learning from text corpora. Nowadays, the ML techniques become more efficient and achieved higher performance by using deep learning.

Deep learning is a subset of machine learning based on artificial neural networks. Deep learning architectures such as feedforward neural networks, recurrent neural networks (RNN) and convolutional neural networks (CNN), have been applied to many fields including but not limited to computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance[Wikipedia, 2020b].

But there are still many limits and challenges in deep learning, such as overfitting, hyperparameter optimization, and sometimes, long training times. Multi-task learning

(MTL), in terms of deep learning, is a broadly-used approach in addressing issues like overfitting and getting better model performance for multi-label tasks.

1.1 Motivation

Even though deep learning is a suitable and powerful approach for addressing natural language processing tasks, training deep learning models on a single task may still not be able to get the best result. Besides overfitting, the information contained in a single label sometimes is not sufficient for the model to "understand" the whole task. Use soccer as an example: to improve his performance, a soccer player not only needs to train on the pitch, but also needs to train weightlifting, marathon, and swimming because these sports can build the muscles which are crucial to them but not easy to be strengthened sufficiently from playing soccer only.

It is the same when it comes to task learning. For a label L, the other labels may contain useful information for L and in the meanwhile training on other labels could prevent the model from overfitting, thereby generating a more general model.

1.2 Contributions

In this thesis, we describe the mechanism and usage of multi-task learning, how it can be combined with deep learning and how to use it in natural language processing tasks.

In experiment 1, the contributions consists of confirming the superiority of multi-task learning over single-task learning on the CL-Aff shared task 2019 dataset, and verifying the effectiveness of pre-trained embeddings, including GloVe and ELMo, on this dataset. Parts of this chapter were published in [Xin and Inkpen, 2019].

In experiment 2, we compare several multi-task deep learning models and contribute a novel way to implement hard parameter sharing using the Venn diagram. In the CL-Aff shared task 2020 competition, our proposed Venn-diagram-based model obtained the best score in the disclosure label group and was very close to the best score in the supportiveness label group. Our paper describing the model [Xin and Inkpen, 2020] was chosen as one of the two the best system description papers.

1.3 Outline

We will first introduce the main types of deep learning techniques and how they can be applied for natural language processing. In chapter Background 2, we will discuss in more detail the mechanisms in the artificial neural network, including artificial neurons, activation functions, and the backpropagation algorithm. Then there is a part focusing on how to connect natural language processing with deep learning, which requires word

embeddings and different types of processing layers in the neural network. Different aspects of multi-task learning are included in the following sections, e.g., the definition of MTL, the motivation behind it, and four important mechanisms in MTL. Then we will discuss how MTL affects the performance on a noisy dataset, and two approaches to employ deep multi-task learning model: hard parameter sharing and soft parameter sharing. According to our experiments, we will only discuss hard parameter sharing in detail in this paper.

Chapter 4 presents several related works on multi-task learning in different NLP fields, such as mental health, emotion analysis, and social media.

The following two chapters are our experiments. Both of them are based on the CL-Aff (Computational Linguistics Affect Understanding) shared tasks in the AffCon (Affective Content Analysis) workshop at the AAAI (Association for the Advancement of Artificial Intelligence) conference. Experiment 1 is based on the shared task from 2019 while experiment 2 is based on the shared task from 2020. The topic of the workshop in 2019 is *In pursuit of happiness*, and in 2020 it is *Get it #OffMyChest*.

The last chapter 6, describes the conclusions of our experiments and proposes directions of future work.

The appendix shows details about the issues we encountered during the practical model employment; techniques and tools we used in the experiments, such as the Python libraries, TensorFlow usage, and the approach we used to generate the ELMo embeddings.

Chapter 2

Background

In this chapter, we will introduce deep learning and show how it can be used for natural language processing. We will dive deeply into the structure of the neural network model, including the neurons, activation function and backpropagation. Then, the approach to connecting deep learning with the natural language processing will be explained, which consists of different word embedding methods and a bottom-up neural network structure. After that, we will focus on multi-task learning, including the motivations and the mechanisms behind it. Two ways of employing multi-task learning in deep learning, hard parameter sharing and soft parameter sharing, will be described in this chapter.

2.1 Deep Learning and Artificial Neural Networks

Deep learning is a network-like model that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Each node in the deep learning model is called an artificial neuron, and all the neurons combined produce the artificial neural network. Deep learning is a type of machine learning in the artificial intelligence field, also deep neural networks.

A neural network contains an input layer, one or more hidden layers, and an output layer, like in Fig. 2.1. The input data will be feed into the input layer first. After flowing through the whole network, the output of the model will be calculated at the output layer. If an update is needed, the difference between the expected output and the actual values will be calculated and propagated back. We mentioned above that neural network is composed of one input layer, one or multiple hidden layers, and one output layer, but in some special cases, it can have multiple input and output layers, too.

In Fig. 2.1, the model contains nine nodes in total. Each node here is an artificial neuron, which is the smallest unit in the neural network model.

As shown in Fig. 2.2, the mechanism of artificial neurons in one layer can be described as follows:

1. **Get connections of input nodes.** The input vector can be showed as $X_m = (x_1, x_2, \ldots, x_n)^T$, in which $x_i (1 \leq \mathrm{i} \leq \mathrm{n})$ is the output value from the ith neuron of the previous layer. The weight matrix can be shown as $W_m = (w_{1m}, w_{2m}, \ldots, w_{nm})^T$, where $W_{im} (1 \leq m \leq n)$ describes the weight between the neuron and the ith input neuron of the previous layer.

2. **Calculate the weighted Sum.** The input vector and the weight matrix are multiplied, then subtracted by the bias vector $\mathbf{b}_\mathrm{m}$ and we can get a median vector, which contains input values for all neurons in this layer. For a specific neuron in the layer, the final input value is one of the scalars in the median vector.

3. **Activate using the activation function.** The final input value obtained from the step 2 will be entered into an activation function, which will be described later in detail. In general, the activation function will transform the input value into a ranged value.

The mathematical expression can be shown by the equations 2.1 and 2.2, in which $\mathrm{f}(\cdot)$ is the activation function of the neuron.

$$u_m = \sum_{j=1}^{n} w_{mj} x_j \tag{2.1}$$

$$y_i = f\left(u_m - b_m\right) = f\left(\sum_{j=1}^{n} w_{mj} x_j - b_m\right) \tag{2.2}$$

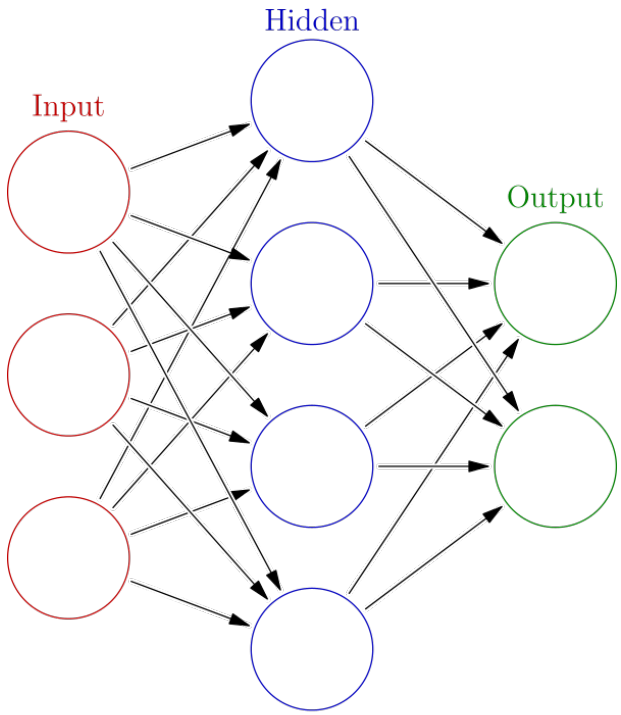


Figure 2.1: Neural network model

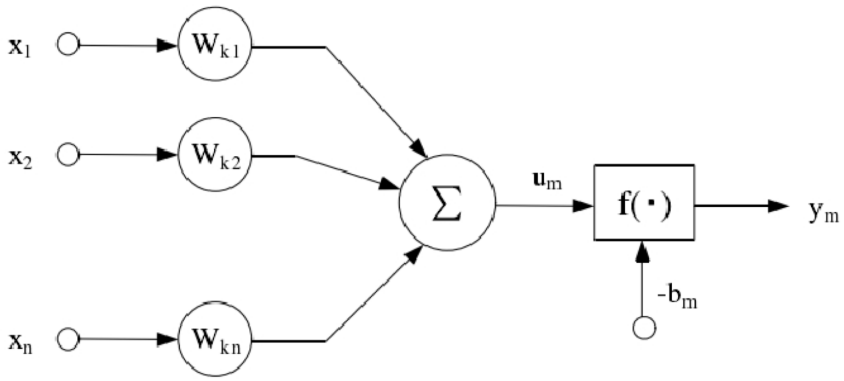


Figure 2.2: The mechanism of an artificial neuron

2.1.1 Backpropagation

So far, we have defined the structure of the neural network. In a newly generated network, all the weights and biases are randomly initialized. Currently, it cannot do any work because all the outputs will be random. To customize the network and make it specific to our task, the neural network needs to be trained. Generally, neural networks are trained via *Gradient Descent*, which has several derivations like: *Stochastic Gradient Descent (SGD)*, *Batch Gradient Descent* and *Mini Batch Gradient Descent*. The mechanism behind gradient descent is called backpropagation.

According to the work from Rumelhart et al. [1986], backpropagation repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the network and the desired output vector. In other words, the aim of backpropagation is to minimize the cost function (loss function) by adjusting the weights and biases of the neural network. The level of adjustment is determined by the gradient of the loss function with respect to the parameters, including the weights and biases.

2.2 Deep Learning Model for Natural Language Processing

The deep learning model used for natural language processing is similar to the original deep learning model we described. They both have an input layer, hidden layers, and an output layer, with weights trained on the training data via backpropagation. The only difference is that the model for natural language processing requires some preprocessing steps for the linguistic data.

Unlike traditional deep learning problems in which the input data are scalar values, in natural language processing, the input data are frequently a set of phrases or sentences in English or any other languages. However, the neural network model can only accept vectors of scalars. So, how to convert the words and sentences into real values is the crucial issue in deep learning for natural language processing tasks.

The approach to mapping words and sentences into vectors of real numbers is called word embedding.

2.2.1 Word Embeddings

Word embeddings are one of the most popular representations of the words in a corpus (collection of documents). They have the ability to capture the context of the words, as well as the semantic and syntactic similarity, relation with other words[1]. In word embeddings, words are represented as vectors of real numbers and these vectors are low-dimensional compared to the vocabulary size. A well-known characteristic here is that words with similar vectors are frequently semantically similar as well.

There are many popular word embedding methods. Among the first ones, there is word2vec which was developed by Mikolov et al. [2013] in 2013 at Google. We do not use word2vec in the following experiments because there are some more efficient successors, like GloVe[Pennington et al., 2014], ELMo[Peters et al., 2018], and BERT[Devlin et al., 2018].

[1]The vocabulary is composed of all the words in the corpus (possibly after removing stopwords and rare words)

- **GloVe** is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations show interesting linear substructures of the word vector space [Pennington et al., 2014].

  As to the usage, we do not always need to train the GloVe vectors by ourselves. There are some pre-trained word vectors released on the official webpage of GloVe. We will use the one trained on Twitter data (*glove.twitter.27B*). It is trained on 2 billion tweets, containing 27 billion tokens; it has a 1.2 million vocabulary size. As most of our experiments are based on the social media data gathered from Facebook or Twitter, it is the most appropriate one for our tasks.

- **ELMo** is a deep contextualized word representation that models two aspects. The first one is the complex characteristics of word usage, for example, syntax and semantics; and the second is how these usages vary across linguistic contexts, for example, to model polysemy[2]. The word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. They can be easily added to existing models and could significantly improve the state-of-the-art results across a broad range of challenging NLP problems, including question answering, textual entailment and sentiment analysis [Peters et al., 2018].

  According to the official webpage[3], ElMo has the following features:

  1. *Contextual*: The representation for each word depends on the entire context in which it is used.
  2. *Deep*: The word representations combine all layers of a pre-trained deep neural network.
  3. *Character based*: ELMo representations are purely character-based, because it contains a character-based context-insensitive type layer, which allows the network to use morphological clues to form robust representations for out-of-vocabulary tokens unseen in training. However, in practice, it can give embedding of anything you put in – characters, words, sentences, paragraphs, even though it is built for sentence embeddings.

  About the usage of ELMo, like GloVe, ELMo also has pre-trained word vectors, which are released both on its official webpage and built in the Tensorflow library. Because it will take a long time to train a new ELMo model, whereas it will not improve the performance too much, we always use the fine-tuned pre-trained models.

  As ELMo has a good balance between performance and training costs, it will be the main approach in the following experiment 1. Unlike GloVe, which only contains a word-value map in its pre-trained model, the use of ELMo is more complex and will be shown in detail in the appendix.

[2] words that have more than one meaning
[3] https://allennlp.org/elmo

- **BERT** (Bidirectional Encoder Representations from Transformers)[Devlin et al., 2018] is published by researchers at Google AI Language in 2018. It was shown to achieve state-of-the-art results in a wide variety of NLP tasks, including Question Answering, Natural Language Inference, and others.

  BERT makes use of the Transformer architecture, an attention mechanism that learns contextual relations between words (or sub-words) in a text. It is pre-trained on a large corpus of sentences. In brief, the training is done by masking a few words in a sentence and tasking the model to predict the masked words. And as the model trains to predict, it learns to produce a powerful internal representation of words as word embedding.

2.2.2 Neural Network Model Structure

In this section, we will describe the structure of deep neural networks for natural language processing, together with the usage of word embedding layers.

Generally from the bottom to the top of a neural network model, we introduce the input layer, embedding layer, convolutional layer, batch normalization layer, pooling and dropping layer, general dense layer for flattening, and output layer.

Input Layer

There is almost no special feature in the input layer. The only thing we should be aware of is that the shape of the input layer can be changed depending on the type of embedding layer we chose.

For example, if we choose GloVe as the embedding layer, the shape of the input layer should be a constant value, and generally, it will be a pre-set length generated from the input tokens. A common way to choose a convenient pre-set length is to find the shortest length, which is no less than 95% [Wikipedia, 2020a] of the lengths of the original sentences. Then the original sentences require cutting if they are longer than the pre-set length or padding if they are shorter than the pre-set length. On the other hand, if ELMo is chosen as the embedding layer, depending on different input requirements for ELMo, the shape can be 1 or a pre-set length from the input tokens.

Embedding Layer

Above next to the input layer is the embedding layer, which can convert the input tokens into numeric values. As we have already talked about different embedding methods before, we will not discuss them again here.

For the balance between performance and training time, loading the pre-trained embeddings is the best choice. But as the pre-trained embeddings were trained on another

corpus, they are not a perfect fit for other datasets. However, almost all embedding methods support updating the embedding values during training. Generally, setting the flag that indicates whether or not updating the values during training is allowed, can improve the model performance by 2 percent, compared to the non-updating model.

Convolutional Layer

Nowadays, convolutional neural networks (CNNs) are among the most popular classifiers in deep learning, especially in image recognition and image classifications, which are used in object detection, face recognition, etc. Although the primary usage of the convolution neural networks is for images, it also has state-of-the-art performance in natural language processing[Al-Ajlan and Ykhlef, 2018] [Husseini Orabi et al., 2018] [Collobert et al., 2011].

The convolutional layer is the core of a convolutional neural network. It carries most of the computational load within the network. The convolutional layer performs a dot product between two matrices, where one matrix is a set of trainable parameters, which is also known as a kernel or a filter, and the other one is the restricted portion of the receptive field.

The kernel slides across the height and width of an image and produces the image representation of that receptive field during the forward pass. The resulting output is called a feature map or activation map. The sliding size of the kernel is called a stride. The convolution operation can be seen in Fig. 2.3

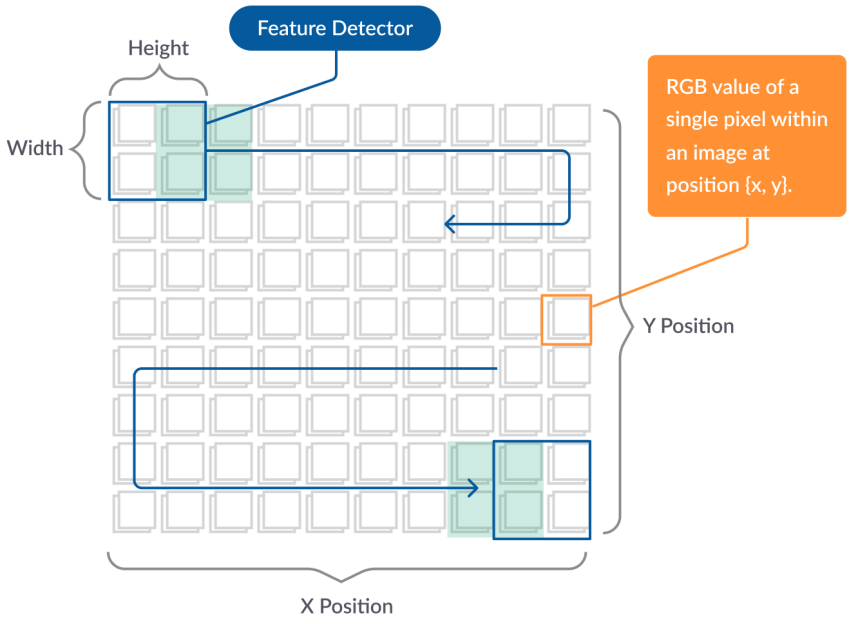


Figure 2.3: An example of 2-D convolution

Layers that perform the convolution operation are collectively known as the convolutional layer. There are many subtypes in this category, such as the 1-D convolutional

layer (e.g., temporal convolution), 2-D convolutional layer (e.g., spatial convolution over images), and so on. Between these two, the 1-D convolutional layer is used in natural language processing, because it scans from the beginning of a text till its end, just as humans read texts.

Unlike general 2-D CNN models, which accept a two-dimensional input representing pixels and colour channels of an image, the 1-D convolutional layer extracts features from one-dimensional sequences of data and maps the internal features of the sequence. In the 1-D convolutional layer, the filter moves from one side of the input data to the other directly without changing direction. For example, in Fig. 2.4, the length of the input sentence, ***I love one dimensional convolutional neural networks very much***, is 9. And each word in the sentence has been mapped to an embedding with a length of 6. The filter, aka kernel, of the 1-D convolutional layer is six units long and two units wide. This allows it to scan the input data from side to side perfectly, just like a process where a human reads a sentence.

In general, the convolutional layers in convolutional neural networks can summarize the features in the input. However, the summarizing ability of convolutional layers is so strong and sensitive, which can cause an issue: the output feature maps are way too sensitive to the location of the features in the input[François, 2018]. In other words, any small movement of the features in the input will result in different feature maps. Using image input as an example, any re-cropping, rotation, shifting, or other minor changes to the image will generate totally different feature maps. As to natural language text input, the issue occurs when switching the order of the words in the sequence, replacing one word to another, etc. In order to address this issue, a widely used technique is downsampling [François, 2018]. There are two common ways to employ downsampling: the first one is changing the strides of the convolutional layer. The stride is the number of pixels shifts over the input matrix. When the stride is 1, then we move the filters to 1 pixel at a time. When the stride is 2, then we move the filters to 2 pixels at a time, and so on. Fig. 2.5 shows how convolution works with a stride of 2. Another approach is to use a pooling layer 2.2.2.

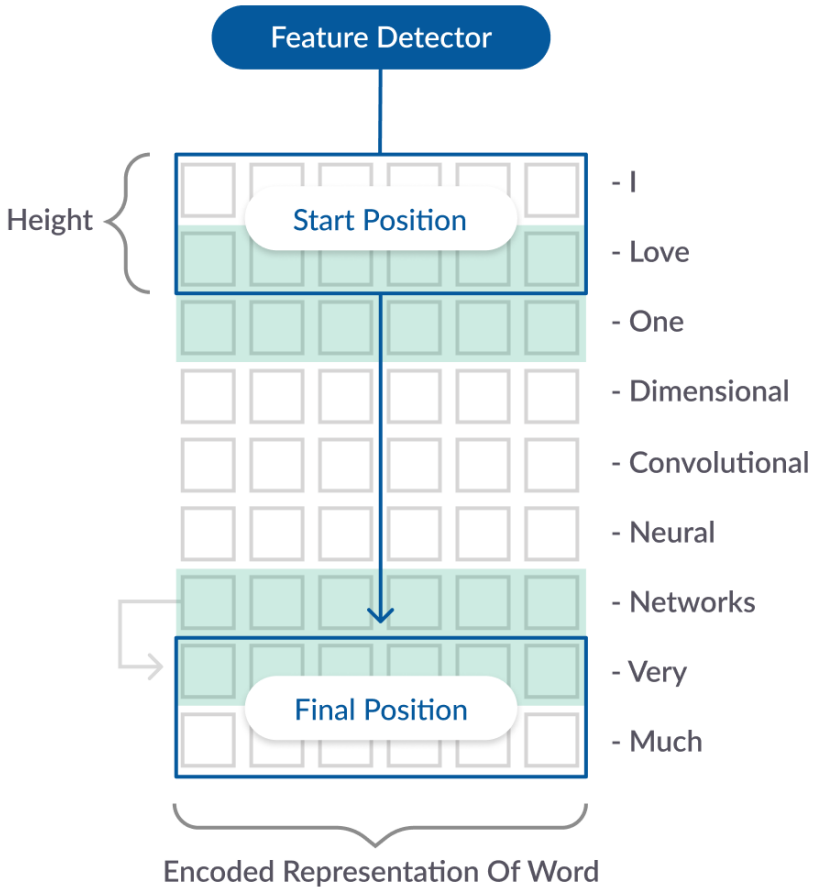


Figure 2.4: An example of 1-D convolution.

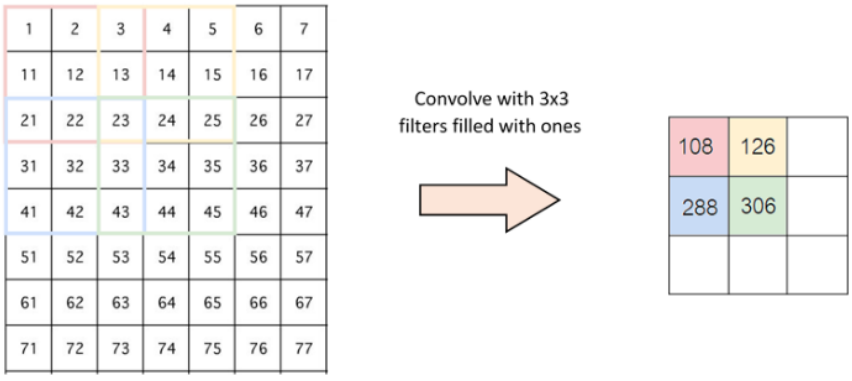


Figure 2.5: An example of Stride 2.

Batch Normalization Layer

In a dataset, the distribution between training data and test data can be different, which causes an issue called covariate shift [Ioffe and Szegedy, 2015]. Normally, we would expect them to have the same distribution, but this rarely ever happens. For example, assuming

we trained a model on images of black cats only, if we now try to apply it on a dataset of coloured cats, it is obvious that the network will not work well because the distribution on test data has changed. The batch normalization layer can improve the performance of the model by reducing the effect of the covariate shift.

Normalization is a method that can normalize the data into a proper scaling. For example, we have some features range from 0 to 1 and some from 0 to 1000. We should normalize them to speed up training. Just as we do this on the input layer by adjusting the scaling of the input, batch normalization can do similar things on hidden layers and get more performance improvement.

To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch's standard deviation [Ioffe and Szegedy, 2015]. A batch normalization layer can normalize the activations of the previous layer at each batch. In other words, it applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Pooling and Dropping Layer

Pooling layers offer an approach to downsampling feature maps by summarizing the presence of features in patches of the feature map. As we said before, pooling layers can help the convolutional layer overcome the limitation that the feature map output records the precise position of features in the input.

Average pooling and max pooling are two common functions used in the pooling operation, which summarize the average presence of a feature and the most activated presence of a feature, respectively.

- **Average Pooling**: Calculate the average value for each patch on the feature map.
- **Maximum Pooling (or Max Pooling)**: Calculate the maximum value for each patch of the feature map.

Besides the pooling method, another parameter that needs to be specified is the filter size, which needs to be smaller than the size of the output of the previous layer. E.g. if the pooling layer is added after the convolutional layer, the filter size should be smaller than the size of the feature map. Specifically, it is almost always 2×2 pixels applied with a stride of 2 units, which means that the pooling layer will always scale down the size by a factor of 2. For example, a pooling layer applied to a feature map of 6×6 will result in an output of size 3×3.

As there is no additional parameter or variable in the pooling layer, we can see that the pooling operation is specified, rather than learned.

There is another type of pooling operation that is sometimes used called global pooling. If we take a feature map as an example, instead of downsampling small patches of the feature map, global pooling downsamples the entire feature map into a single value. It is a way that aggressively summarizes the presence of a feature.

Activation function

Activation functions play a key role in neural networks. They are applied to every neuron and allow the neural network to learn non-linear states. If the activation function is not applied, the output signal of each neuron will become a simple linear function. A non-activated neural network will act as a linear regression with limited learning power, compared to the complex input information, like sentences, images, or sounds.

There are many popular activation functions available nowadays. The choice of activation functions in deep neural networks has a significant impact on the training dynamics and task performance. Here we will describe the sigmoid function, the ReLU function, the Leaky-ReLU function and the Swish function.

- **Sigmoid Function** is the most popular and frequently used activation function, shown in the Fig. 2.6. The mathematical expression of the sigmoid function is:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

  One advantage of the sigmoid function is that the value of it ranges from 0 to 1, which means it can still produce an output in the range of (0,1) even when encountered with (- infinite, + infinite) as input and the activation value will not vanish. Another advantage is that as the value tends to 1 along the direction of the positive x-axis, and to 0 along the direction of the negative x-axis. The sigmoid function can also be used as a binary classification function.

  However, the sigmoid function has a crucial disadvantage. If we look carefully at the graph, we can find out that the derivative values at 1 and before -1 are very small and converge to 0. This causes a problem called vanishing gradient and harms the learning process. It is easy to imagine that in a deep neural network, where each layer produces a small derivative value and passes it to the next layer, the gradient will disappear quickly, and the difference between outputs and true values cannot be delivered to the bottom layers. In order to address this problem, many other more efficient alternatives appeared, like ReLU.

- **ReLU(Rectified Linear Unit)** is the most successful and widely-used activation function. The mathematical expression of ReLU is:

$$f(x) = \max(0, x) \tag{2.4}$$

  Its value ranges from [0, +infinite). As shown in Fig. 2.7, the most prominent feature of ReLU is that its derivative value remains 0 at the negative x-axis, and 1 at the positive x-axis. Keeping the derivative value at 1 on the positive axis solves the vanishing gradient problem effectively.

  But for the negative axis, it has a value of 0, which means it can provide a more efficient computational load in the network, and the model can run faster. However,

as both the output and the derivative value are 0, the model cannot learn on this zero value region, which leads to a lot of dead neurons in the network model. It is a problem called *dead ReLU*. This disadvantage outweighs its merits, and this is why LeakyReLU appeared with a trick.

- **LeakyReLU** has a small leak on the negative plane, compared to ReLU, as shown in Fig. 2.8. The mathematical expression is:

$$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.5}$$

  The derivative of the LeakyReLU is still 1 in the positive part and is a small, constant fraction in the negative part.

  However, even though LeakyReLU has its advantage, it is at the cost of computational power. And as ReLU has been broadly applied in most evaluation criterion models, most people still choose ReLU as the activation function. It is not a bad choice. Most of the time, leaving dead neurons dead will not harm the model performance. So leaky or not leaky ReLU, it still depends on what a model needs.

- **PReLU** extends the idea of LeakyReLU by using the model itself to train the small leak value. If the model can learn that small value during training, the activation function can better adapt to the other parameters like weights and biases. The slope parameter is learned using backpropagation at a negligible increase in the cost of training. The mathematical expression of PReLU is:

$$f(x) = \begin{cases} a_i x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.6}$$

- **Swish** [Ramachandran et al., 2017] is a smooth, non-monotonic function, as shown in Fig. 2.9. The standard mathematical expression of Swish is:

$$f(x) = x \cdot \text{sigmoid}(x) \tag{2.7}$$

  Swish is unbounded above and bounded below, which, together with its non-monotonic attribute, actually creates the difference. All other activation functions are monotonous, but the output of the Swish function may even fall when the input increases, which is a swish-specific feature.

  About the performance, Swish consistently matches or outperforms ReLU on deep networks applied to a variety of challenging domains such as image classification and machine translation according to Ramachandran et al. [2017]. There are some other advantages, like the simplicity of Swish and its similarity to ReLU, which makes it easy for practitioners to replace ReLUs with Swish units in any neural network.

Dense Layer and Output Layer

In neural networks, the dense layer is the most simple layer in which each neuron fully connects to all neurons in the previous layer. Here, we mention the dense layer and the output layer because, in general, each neuron network model only has one output layer, but in hard parameter sharing multi-task learning, a neural network model can have multiple output layers.

In our model, above the dropping and pooling layer, the network connects to multiple dense layers at the same level. The number of dense layers in this level is equal to the number of tasks. Then each dense layer connects to its task-specific output layer.

The details of this part will be explained later.

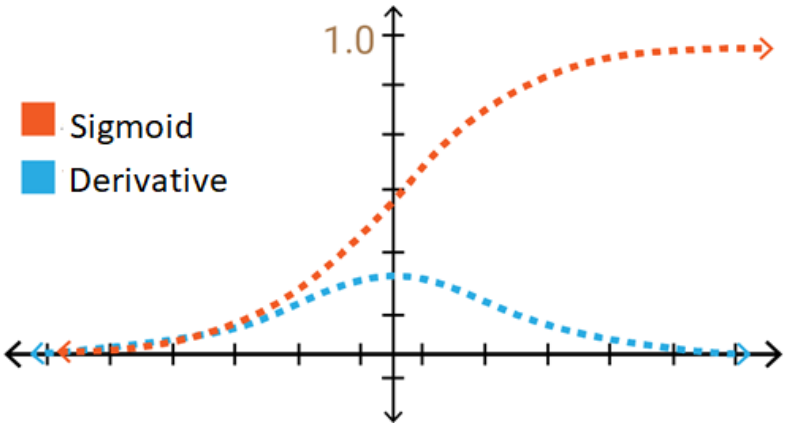


Figure 2.6: Sigmoid Function and Derivative

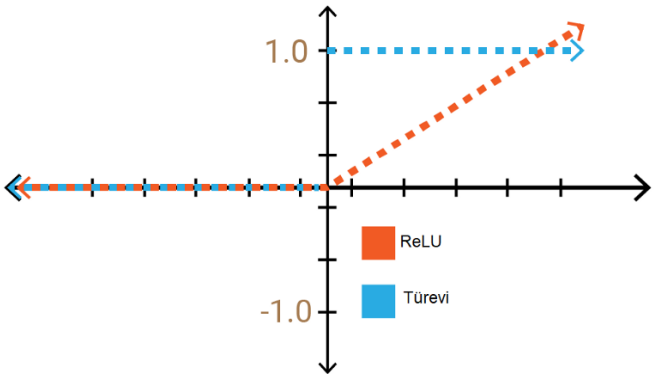


Figure 2.7: ReLU Function and Derivative

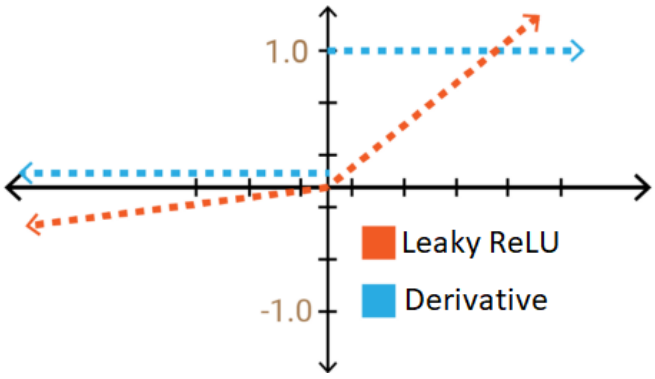


Figure 2.8: Leaky ReLU Function and Derivative

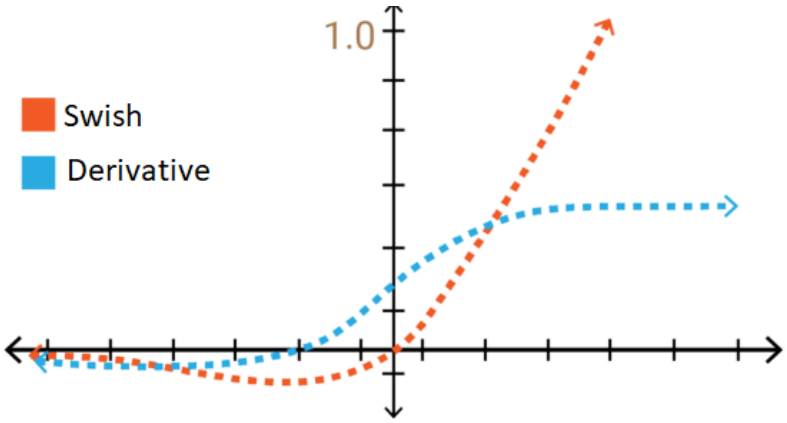


Figure 2.9: Swish Function and Derivative

2.3 Definition of Multi-Task Learning

Intuitively, when we say *multi-tasking*, we image a person who tries to do a lot of different works at the same time. This term always gives us a negative image, because, from another perspective, it means the person does not *focus* on one thing. But is it really a bad thing? Given that our brain can be considered as a deep neural network model if we think about our vision system, what is the output of that? When we observe a street, we will not only identify the street itself, but also see the people on the street, roadside buildings, and the colour of the sky. Another example: when we listen to words from a person, we can not only understand the sentence he/she speaks and its meaning, but also know the sex of the person, and his/her current mood, at least partially. So, from a human perspective, the brain is always a multi-tasking system.

Multi-task learning aims to learn multiple different tasks simultaneously while maximizing performance on one or all of the tasks. Generally, when you train more than one loss function, or a part of your loss function comes from another task, you are doing multi-task learning.

The definition of multi-task learning (MTL) given by *Zhang and Yang*[Zhang and Yang,

2017] is:

Definition 1 ***(Multi-Task Learning)*** *Given m learning tasks $\{\mathcal{T}_i\}_{i=1}^m$ where all the tasks or a subset of them are related, multi-task learning aims to help improve the learning of a model for $\mathcal{T}_i$ by using the knowledge contained in all or some of the m tasks.*

MTL has many aliases: joint learning, learning to learn, and learning with auxiliary tasks are refer to the same thing [Ruder, 2017].

In classification problems, there are two concepts that are easily confused: multi-class classification and multi-label classification. They all have a prefix of "multi", so are both of them referring to multi-task learning?

Multi-class classification [Wikipedia, 2020e] means a classification task with more than two classes; e.g., classify a set of images of animals which may be dogs, cats, or hamsters. Multi-class classification makes the assumption that each sample is assigned to one and only one label: an animal can be either a cat or a dog, but not both at the same time. So classes are mutually exclusive.

Multi-label classification [Wikipedia, 2020d] assigns to each sample a set of target labels. This can be thought of as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a text. A tweet might be about any of religion, politics, emotion or personal information at the same time or none of these.

Theoretically speaking, both of the multi-class and multi-label classification tasks can be converted into multi-task problems. For a multi-class classification task, which contains three classes: dog, cat and hamster, each class in this label can be picked up and treated separately. Then a multi-class classification task will be converted into a three-label binary classification problem and can be adapted to a multi-task learning framework. But, as the three labels have a strong relationship, adding constraints into the MTL framework might be required.

Nevertheless, most of the time, when we say multi-task learning, we refer to a multi-label classification problem. If all labels in the task are binary, it will be a multi-label binary classification problem. Both of the two experiments in chapters 5 and 6 are multi-label binary classification tasks.

2.4 Motivation

In order to solve a large problem, the standard way is to break it into small, independent subproblems and then learn them separately. But sometimes, this methodology is counterproductive because it may ignore a potentially rich source of information contained in other tasks.

Biologically, when humans learn something new, they prefer to use the knowledge they already learned from other things. For example, if you are the first time playing baseball,

you can still get help from the knowledge about how to move your body from other sports you already learned.

From the view of machine learning, multi-task learning can be treated as a special form of inductive transfer[Ruder, 2017]. In inductive transfer, model performance is improved by the introduction of extra inductive biases, which lead the model to prefer some hypotheses. For instance, L1 regularization is a common form of inductive bias, which makes the model prefer some sparse results. In multi-task learning, inductive biases are provided by auxiliary tasks, which makes the model prefer to generalize to all the tasks instead of only one task.

2.5 Mechanisms in Multi-Task Learning

We know that the improved performance of multi-task learning is due to the extra information in related tasks, but how does this occur? According to the work from Caruana [1997], it can come down to four mechanisms.

2.5.1 Statistical Data Amplification

Data amplification is an *effective* increase in sample size due to extra information in the training signals of related tasks [Caruana, 1997]. Amplification occurs when there is noise in the training data [Caruana, 1997]. For example, we have two tasks *T1* and *T2*, with independent noise added to their training signals, and two neural network models *M1* and *M2*, which both have only one hidden layer *F1* and *F2*, computed from *T1* and *T2* separately. Compared with these two networks, the hidden layer F in another neural network model *M3*, which is trained on both *T1* and *T2* simultaneously, can be learned better by averaging the values for F through the different noise patterns.

In other words, for a task T, even though the amount of data seems unchanged, apparently, by adding an auxiliary task T', it still benefits from the hidden information or noise in T' during training; this is called data amplification.

2.5.2 Attribute Selection

Attribute selection is a consequence of data amplification [Caruana, 1997]. Consider two tasks, T and T', which use a common hidden layer F. Supposing there are many inputs to the network and the data is limited or significantly noisy, the network may have difficulty distinguishing features in the inputs which are really relevant to F. A network learning both T and T' can better select attributes relevant to F because the data amplification provides better training signals for F, allowing it to better determine which input to use during training.

2.5.3 Eavesdropping

Consider a feature E which is important for both task T and task T', and it is easy to be learned in task T, but difficult in T' (which may due to T' uses a more complex way to learn E or the bottom layer in T' is noisier). A network learning T will learn E, but a network learning T' may not. However, if a network learning both T and T', T' can learn better by eavesdropping feature E learned from task T.

2.5.4 Representation Bias

We know that the weights and biases in neural networks are updated by backpropagation, which in practice is trained using stochastic gradient descent (SGD). The aim of SGD is to find the global minimum, but most of the time, it will convergence at some local minimum. In multi-task learning, the search is biased towards representation in the intersection of what would be learned alone, as shown in Fig. 2.10, which yields two results: MTL tasks prefer hidden layer representations that other tasks prefer, and prefer NOT to use hidden layer representations that other tasks prefer NOT to use [Caruana, 1997].

These two conclusions are proved by Caruana [1997] in two experiments. In the first experiment, the networks trained on task T alone are equally likely to find local minima A or B, while networks trained on task T' are equally likely to find local minima A or C. Then they find that networks trained on both task T and T' usually fall into A for both tasks. In the second experiment, networks trained on task T has a strong preference for the local minimum B over A, while on T' has equally preference between local minima A and C. Surprisingly, the networks trained on both task T and T', T fall into B as expected, but T' usually fall into C.

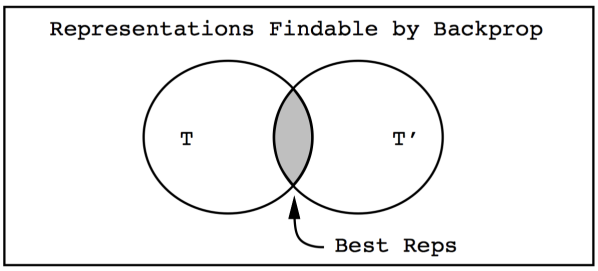


Figure 2.10: Representation Bias

2.6 How Multi-Task Learning Works on Noisy Data

All datasets are somewhat noisy. When training a model for a task, our aim is to do the best, ignoring the data-dependent noise and learning a good pattern on other features. Because different tasks have different noise patterns, a model trained on multiple tasks is able to generate a more general representation and average the noise patterns on different

tasks. If there is a very noisy dataset and the model is in trouble distinguishing relevant and irrelevant features, auxiliary tasks can help model focus on the features that really matter because they provide additional evidence about whether the feature is relevant or not [Ruder, 2017].

2.7 Multi-task Learning in Deep Learning

In the context of Deep Learning, multi-task learning is typically done with either hard or soft parameter sharing of hidden layers [Ruder, 2017].

2.7.1 Hard Parameter Sharing

Hard parameter sharing is the most commonly used method in multi-task deep learning. The concepts behind it is simple. As shown in the figure 2.11, some hidden layers are shared across all tasks, while each task can still have its own task-specific layer. In this way, the parameters in the shared hidden layer are forcibly generalized to all tasks, having a lower risk of overfitting on a specific task. That is to say, the more tasks we are learning simultaneously, the more our model will find a representation that captures all of the tasks and the lower is our chance of overfitting on our original task.

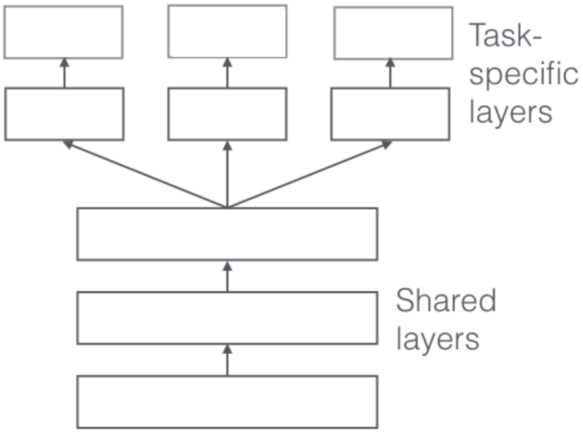


Figure 2.11: Hard Parameter Sharing Model

2.7.2 Soft Parameter Sharing

Soft parameter sharing, on the other hand, will not share layers between tasks. Instead, it will add constraints between layers in different tasks. As shown in Fig. 2.12, the distance between parameters is regularized in order to encourage them to be similar. This was inspired by the regularization techniques in machine learning and had many different forms.

For instance, in Fig. 2.12, it uses ℓ_2 norm for regularization, while in 3.3, the author uses trace norm.

In other words, in soft parameter sharing, each model has its own sets of weights and biases, while the distance between these parameters in different models is regularized so that the parameters become similar and can represent all the tasks.

Unlike hard parameter sharing, soft parameter sharing does not have a classic structure. The constraints put on the network should be decided by the users themselves according to the task requirements and model structures. Because of the high subjectivity and task dependency, we will not use soft parameter sharing in the following experiments.

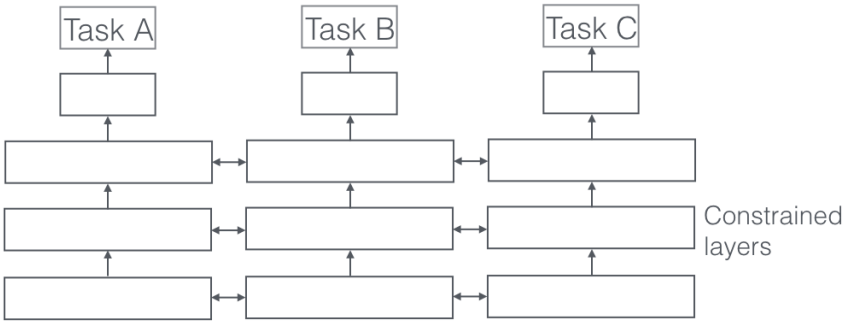


Figure 2.12: Soft Parameter Sharing Model

Chapter 3

Related Work on Multi-Task Deep Learning

3.1 Multi-Task Learning for Mental Health Using Social Media Text

The paper [Benton et al., 2017] published by Benton et al. [2017] introduces initial groundwork for estimating suicide risk and mental health issues in a deep learning framework. The system learns to make predictions about suicide risk and mental health by modelling multiple conditions as multiple tasks in a multi-task learning framework, together with gender prediction as an additional auxiliary task. To evaluate the multi-task learning method, the author compares the MTL model against single-task models with the same number of parameters, finding that the MTL model increases the true positive rate and modelling gender can improve accuracy across a variety of conditions.

According to [Caruana, 1993b] and some other works, e.g.[Caruana et al., 1996] and [Lipton et al., 2016], MTL approaches have shown considerable improvements over single-task models. The argument is convincing: predicting multiple related tasks should allow the model to exploit any correlations between the tasks. However, the authors found that in their work, the MTL model is only one possible explanation for the improved accuracy, while another more salient factor could have been overlooked most of the time: the difference between model expressiveness, i.e., differences in the number of parameters for comparable models.

In order to examine the effect of multi-task learning specifically, a standard logistic regression model, a multi-layer perception single-task learning (STL) model, and a neural MTL model was deployed [Coppersmith et al., 2015] on the difficult task of predicting mental health conditions from social media (user-generated texts). The single-task model and multi-task model are shown in Fig. 3.1. To ensure a fair comparison, the two models have an equal number of parameters.

As shown in the left side of Fig. 3.1, STL models are feedforward networks with two hidden layers, while the right side in Fig. 3.1 shows the MTL model, where the first

hidden layer from the bottom of the network is shared between all tasks. An additional task-specific hidden layer is used for each task to give the model flexibility to map from the task-agnostic representation to a task-specific one. Rectified linear units are used in each hidden layer as non-linear activation functions. The output layer uses a logistic non-linearity to produce binary predictions for all tasks.

The authors use mini-batch in the training because it allows optimization to jump out of poor local optima and more stochastic gradient steps in a fixed amount of time, as noted in [Collobert et al., 2011] and [Bottou, 2012].

As to the results, Fig. 3.2 shows the AUC-score of each model for each task separately, and Fig. 3.3 shows the true positive rate at a low false-positive rate of 0.1. Both the AUC and TPR values demonstrate that single-task models do not perform well compared with multi-task models, or even logistic regression models. The reason behind this, according to the authors, is that MTL may provide a form of regularization that STL cannot exploit, and further, the MTL model can help in predicting elusive conditions by using large data for the common conditions and a small amount of data for the rare conditions. And both Fig. 3.2 and Fig. 3.3 suggest that adding gender as an auxiliary task leads to better predictive models.

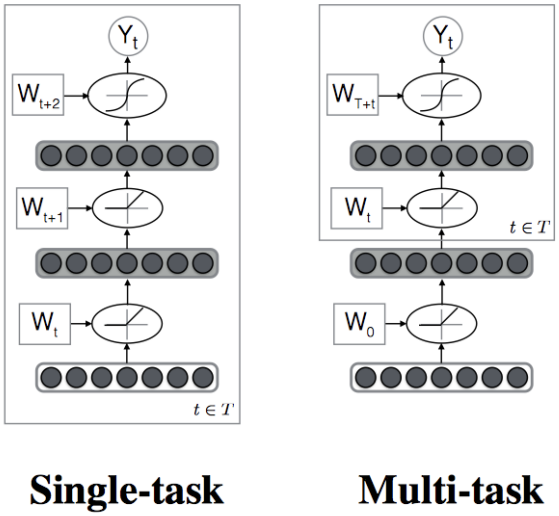


Figure 3.1: STL model(left): weights trained independently for each task t; MTL model (right): shared weights trained jointly for all tasks, with task-specific hidden layers.

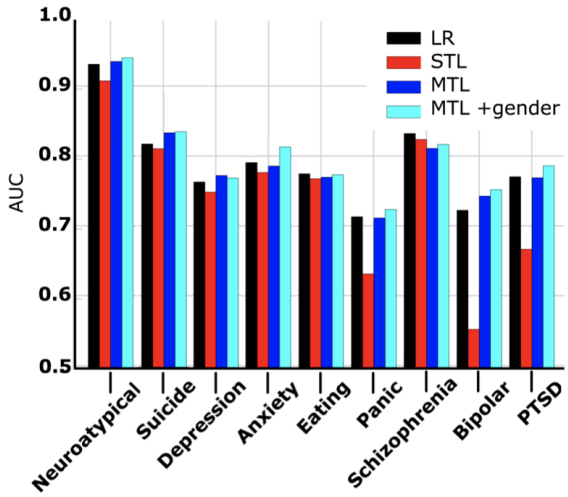


Figure 3.2: AUC for different tasks

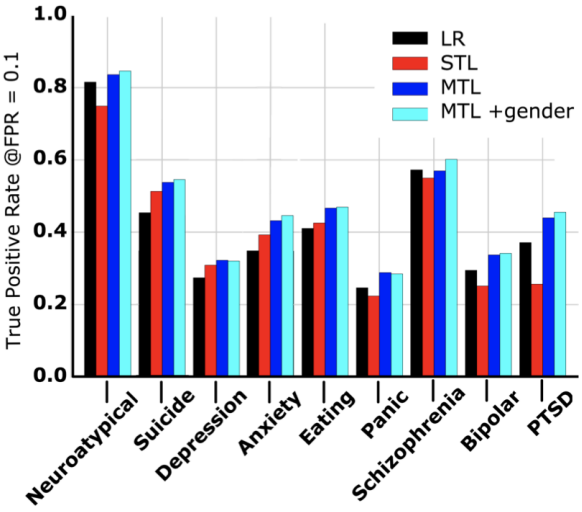


Figure 3.3: TPR at 0.10 FPR for different tasks

3.2 Emotion-Cause Pair Extraction

The main task in the paper [Xia and Ding, 2019] is emotion cause extraction (ECE), which aims at extracting potential causes that lead to emotion expressions in the text. In this paper, the authors Xia and Ding [2019] propose a new task called *emotion-cause pair extraction (ECPE)*, which aims to extract the potential pairs of emotions and corresponding causes in a document. The approach uses two-steps to address the ECPE task. It applies multi-task learning; this is why we mention this paper here.

First of all, an example of ECPE and the difference between ECE and ECPE is shown in the Fig. 3.6. The definition of ECPE can be described as the following: given a document

document consisting of multiple clauses $d = \left[c_1, c_2, \ldots, c_{|d|}\right]$, the goal of ECPE is to extract a set of emotion-cause pairs in d :

$$P = \{\cdots, (c^e, c^c), \cdots\}$$

where c^e is an emotion clause and c^c is the corresponding cause clause.

The concept of the two-step approach addressing the ECPE task is simple:

- **Step 1 Individual Emotion and Cause Extraction** The emotion-cause pair extraction task is converted into two individual sub-tasks: emotion extraction and cause extraction. Then two kinds of multi-task learning networks are proposed to model the two sub-tasks in a unified framework. This step is what we will focus on later.
- **Step 2 Emotion-Cause Pairing and Filtering** A Cartesian product is applied to the emotion set E and the cause set C, which yields a set of candidate emotion-cause pairs. Then a filter is trained on them to eliminate pairs that do not contain a causal relationship between emotion and cause.

In step 1, two kinds of multi-task learning networks are proposed by the authors, i.e., independent multi-task learning and interactive multi-task learning. The latter captures the correlation between emotion and cause, and it is an enhanced version of the former one.

3.2.1 Independent Multi-Task Learning

The independent multi-Task learning framework is a Hierarchical bi-directional LSTM (bi-LSTM) network that contains two layers, i.e., the brown parts and green part in Fig. 3.4. Here the authors omitted the output layer, i.e., the red part in Fig. 3.4 and the copy operation, i.e. the blue part in Fig. 3.4.

Clause representation is generated by the bi-LSTM and attention mechanism at the lower layer, while the upper layers focus on the extraction. There are two components in the upper layer: one for emotion extraction and another for cause extraction. Each component is a clause-level bi-LSTM which receives the independent clause representations obtained at the lower layer. The loss of the model is a weighted sum of two components:

$$L^p = \lambda L^e + (1 - \lambda) L^c$$

where L^e and L^c are the cross-entropy error of emotion prediction and cause prediction respectively, and λ is a trade-off parameter.

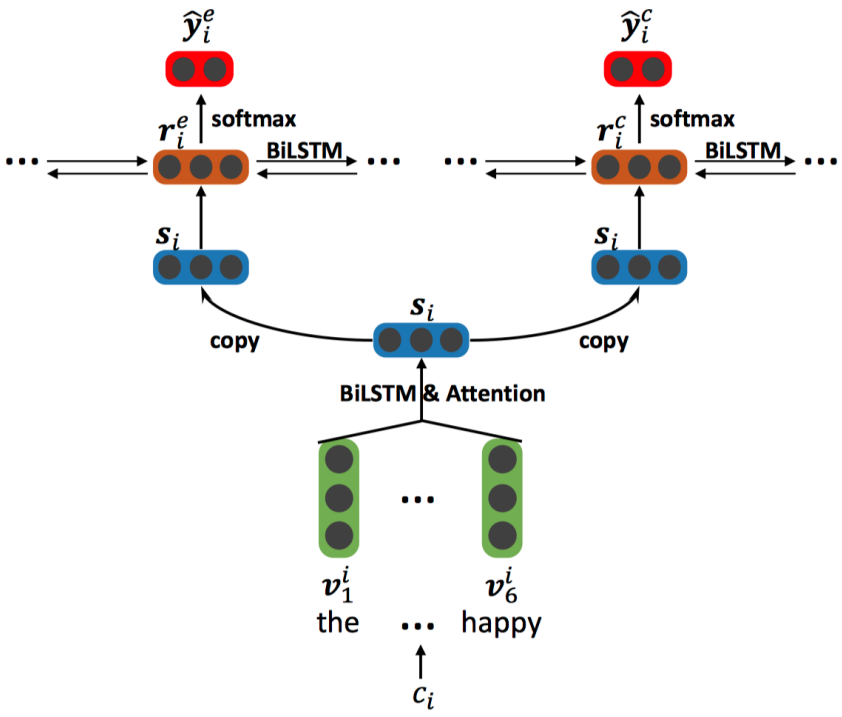


Figure 3.4: The model for Independent Multi-Task Learning.

3.2.2 Interactive Multi-Task Learning

As we saw in the independent multi-task learning, there are two components at the upper layer, which correspond to two sub-tasks: emotion extraction and cause extraction. As these two sub-tasks are not mutually exclusive, keeping the lower layer unchanged, providing results from one of them into the input of the other can help improve the model's performance.

Based on this concept, the interactive multi-task learning framework authors proposed can be shown in Fig. 3.5. Inter-EC refers to using emotion extraction to improve cause extraction, whereas using cause extraction to improve emotion extraction is called Inter-CE.

The loss of the model is the same as in Independent Multi-Task Learning (the weighted sum of two components).

In the evaluation, the authors found that compared with Independent Multi-Task Learning, Inter-EC and Inter-CE gets significant improvements on the ECPE task. This shows that the predictions of the previous task are beneficial to the latter extraction task and prove the effectiveness of the Inter-EC and Inter-CE.

By comparing the results of Inter-EC and Inter-CE, the authors found that the improvement of Inter-EC is mainly obtained on the cause extraction task, and the improvement of Inter-CE is mainly obtained on the emotion extraction task, which is consistent with the authors' hypothesis that emotion and cause are mutually indicative.

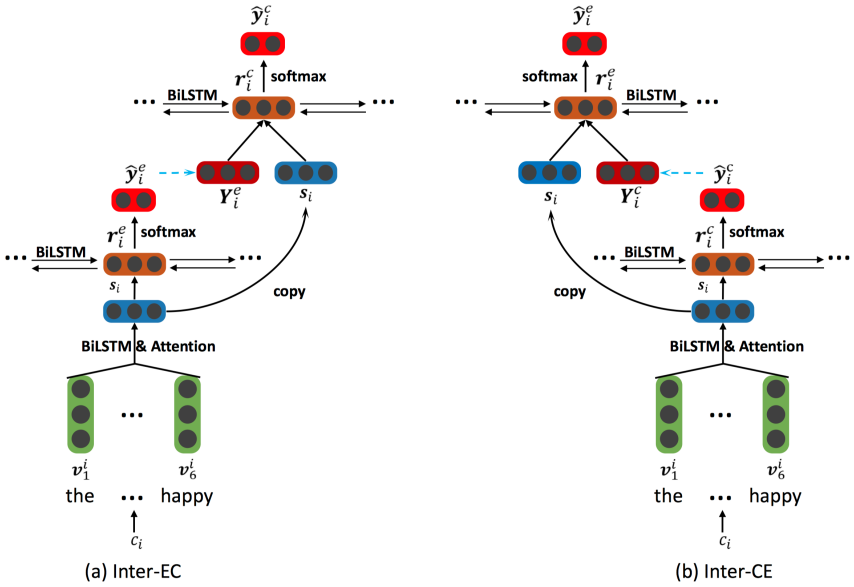


Figure 3.5: Two Models for Interactive Multi-task Learning: (a) Inter-EC, which uses emotion extraction to improve cause extraction (b) Inter-CE, which uses cause extraction to enhance emotion extraction.

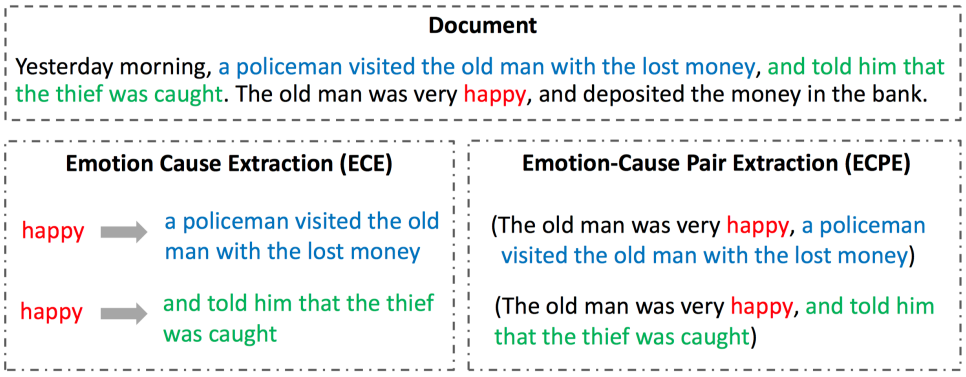


Figure 3.6: An example showing what is ECPE and the difference between ECPE task and ECE task.

3.3 Trace Norm Regularised Multi-Task Deep Learning

This section involves soft parameter sharing in the neural network. The authors of the paper [Yang and Hospedales, 2016] proposed a framework for training multiple neural networks simultaneously. The parameters from all models are regularised by the tensor norm. In this way, the neural networks are encouraged to reuse each other's parameters, if possible.

According to the authors, primarily, the idea is derived from the objective of multi-

task training from Caruana [1997]: MTL aims to learn multiple tasks jointly and reuse knowledge obtained from other tasks. Instead of predefining a parameter sharing strategy, the authors propose a novel framework:

Methodology For T tasks, each task is modelled by a neural network of the same architecture. The parameters are collected in a layer-wise fashion and put a tensor norm on every collection [Yang and Hospedales, 2016].

The method that puts the tensor norm on neural network parameters is based on the concept from *Tensor-based Multi-Task Learning*. For example, to build a restaurant recommendation system, we want a regression model that predicts the scores for different aspects, like food quality, services and environment, by different customers. Then the task is indexed by aspects $\times$ customers. The collection of linear models for all tasks is then a 3-way tensor $\mathcal{W}$ of size $D \times T_1 \times T_2$, where D is the dimension of the feature vector while T_1 and T_2 is the number of aspects and customers respectively. Consequently, the regularization $\Omega(\mathcal{W})$ on the loss function has to be a tensor regulariser [Tomioka et al., 2010]. There are several different tensor regularisers available, for example, the sum of the trace norms on all matricization [Romera-Paredes et al., 2013], and scaled latent trace norm [Yang and Hospedales, 2014].

The authors chose to use the trace norm as the regulariser. In matrix terms, it can be calculated by the sum of a matrix's singular values $\|X\|_* = \sum_{i=1} \sigma_i$. The extension of trace norm from matrix to tensor is not unique; it depends on how we assume the tensor is factorized. In this paper, the authors proposed three tensor trace norm designs:

For an N -way tensor $\mathcal{W}$ of size $D_1 \times D_2 \times \cdots \times D_N$:

- **Last Axis Flattening** $\|\mathcal{W}\|_* = \gamma \left\|\mathcal{W}_{(N)}\right\|_*$
- **Tucker-rank** $\|\mathcal{W}\|_* = \sum_{i=1}^{N} \gamma_i \left\|\mathcal{W}_{(i)}\right\|_*$
- **Tensor Train(TT)-rank** $\|\mathcal{W}\|_* = \sum_{i=1}^{N-1} \gamma_i \left\|\mathcal{W}_{[i]}\right\|_*$

The results of their experiments are shown in Fig. 3.7. As we can see in Fig. 3.7, STL has the lowest training loss, but the worst testing performance, suggesting that the STL model suffers from over-fitting.

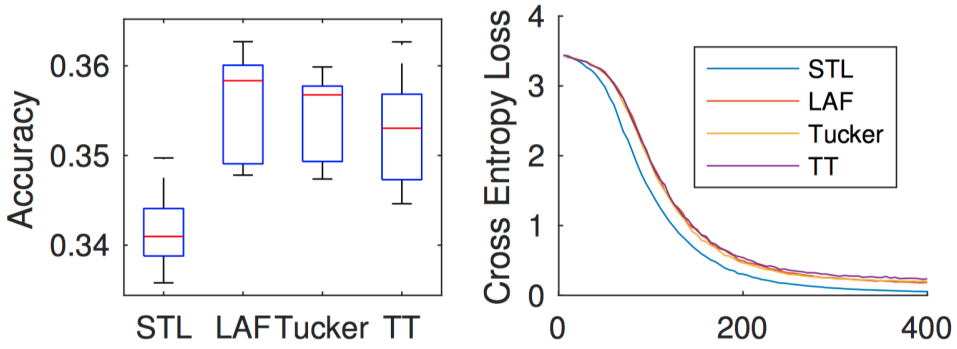


Figure 3.7: Left: Testing accuracy. Right: Training loss

Chapter 4

Experiment 1: Multi-Task Deep Learning for Happiness Ingredients Detection

4.1 Introduction

Happiness is one of the important facets of human emotion. In psychology, it is a certain state of mind. The descriptions of happy moments include events that give satisfaction, pleasure, or a positive emotional condition. For the purpose of natural language processing tasks, it is difficult to formalize happiness. However, as human affect is context-driven, what we are concerned with here is the contextual and agentic attributes of the descriptions of the happy moments.

We participated in the first CL-Aff Shared Task: *CL-Aff Shared Task: In Pursuit of Happiness* [1], which was held as a part of the Affective Content Analysis (AffCon) workshop @ the AAAI 2019 conference. In the quest to understand user expression, the workshop proposed a task focusing on one facet of human affect – happiness. A new labelled corpus of happy moments and two challenges were posted to spur the development of supervised and semi-supervised approaches to model human affect. Two main task labels, *Agency* and *Social*, are introduced in this task. *Agency* describes whether or not the author is in control, while *Social* describes whether or not the moment involves people other than the author. These two labels can help in identifying how subjective behavior and interactions with other people can affect the emotion of a person.

The CL-Aff Shared Task aims to challenge the current understanding of emotion and affect in the text through a task that models the experiential, contextual, and agentic attributes of the crowd-sourced single-sentences that describe happy moments. The Shared Task is organized in collaboration with researchers at Megagon Labs and builds upon the HappyDB dataset [Asai et al., 2018], which is a corpus of more than 100,000 happy moments crowd-sourced via Amazon's Mechanical Turk.

[1] https://sites.google.com/view/affcon2019/cl-aff-shared-task

4.2 Task Description

There are two sub-tasks in the shared task for analyzing happiness and well-being in written language on the modified HappyDB corpus.

- **GIVEN**: An account of a happy moment, marked with an individual's demographics, recollection time and relevant labels.
- **TASK 1: WHAT ARE THE INGREDIENTS FOR HAPPINESS?** Semi-supervised learning task: Predict agency and social labels for happy moments in the test set, based on a small labelled and large unlabelled training data.
- **TASK 2: HOW CAN WE MODEL HAPPINESS?** Unsupervised task: Propose new characterizations and insights (not necessarily and not limited to themes) for happy moments in the test set, e.g., in terms of affect, emotion, participants and content.

We can see that Task 1 is a classification problem that requires predicting agency and social labels (two classes), while task 2 is open-ended, encouraging participants to propose new characterizations and insights for the descriptions of the happy moments in the test set. In this experiment, we will omit task 2 and only focus on task 1.

In task 1, besides two existing tasks (agency and social), we introduced the determination of *Concepts* as the third task, to take full advantage of the power of multi-task learning.

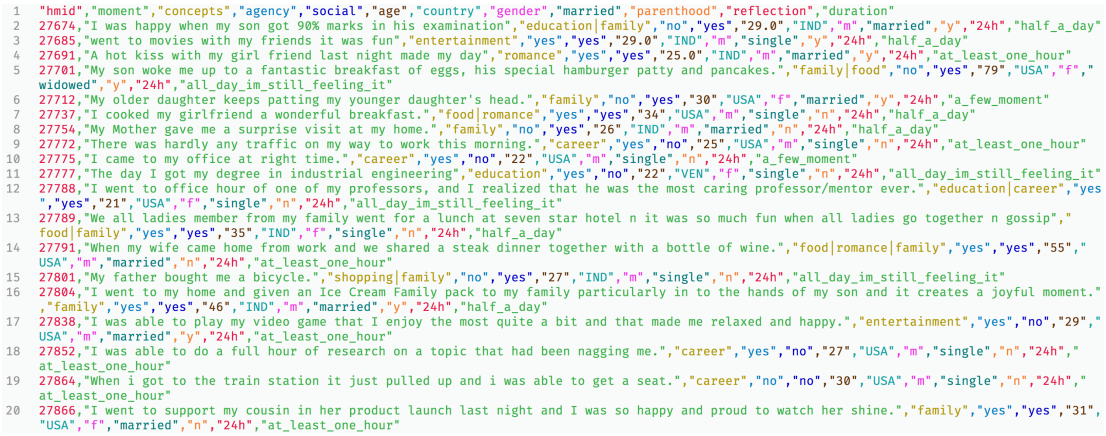
4.3 Data Description

The dataset released in this shared task includes a labelled training set, unlabelled training set and test set.

The labelled training set is composed of single-sentence happy moments from the available HappyDB corpus, annotated with labels that identify the 'agency' of the author and the 'social' characteristic of the moment, as well as concept labels describing its theme. There are 10560 pieces of data in the labelled training set. Each of them has one identification number, one English sentence called *moment* and ten labels including *concepts, agency, social, age, country, gender, married, parenthood, reflection, duration*. Here, we need to pay attention to three labels: *concepts, agency* and *social* only. *Agency* and *Social* are labels related to the task objective, while *concepts* is our auxiliary task in multi-task learning model.

An overview of the dataset is shown in Fig. 4.1.

- *Agency*: Binary label describing whether or not the author is in control. (true/false)



```
"hmid","moment","concepts","agency","social","age","country","gender","married","parenthood","reflection","duration"
27674,"I was happy when my son got 90% marks in his examination","education|family","no","yes","29.0","IND","m","married","y","24h","half_a_day"
27685,"went to movies with my friends it was fun","entertainment","yes","yes","29.0","IND","m","single","y","24h","half_a_day"
27691,"A hot kiss with my girl friend last night made my day","romance","yes","yes","25.0","IND","m","married","y","24h","at_least_one_hour"
27701,"My son woke me up to a fantastic breakfast of eggs, his special hamburger patty and pancakes.","family|food","no","yes","79","USA","f","
widowed","y","24h","all_day_im_still_feeling_it"
27712,"My older daughter keeps patting my younger daughter's head.","family","no","yes","30","USA","f","married","y","24h","a_few_moment"
27737,"I cooked my girlfriend a wonderful breakfast.","food|romance","yes","yes","34","USA","m","single","n","24h","half_a_day"
27754,"My Mother gave me a surprise visit at my home.","family","no","yes","26","IND","m","married","n","24h","half_a_day"
27772,"There was hardly any traffic on my way to work this morning.","career","yes","no","25","USA","m","single","n","24h","at_least_one_hour"
27775,"I came to my office at right time.","career","yes","no","22","USA","m","single","n","24h","a_few_moment"
27777,"The day I got my degree in industrial engineering","education","yes","no","22","VEN","f","single","n","24h","all_day_im_still_feeling_it"
27788,"I went to office hour of one of my professors, and I realized that he was the most caring professor/mentor ever.","education|career","yes
","yes","21","USA","f","single","n","24h","all_day_im_still_feeling_it"
27789,"We all ladies member from my family went for a lunch at seven star hotel n it was so much fun when all ladies go together n gossip","
food|family","yes","yes","35","IND","f","single","n","24h","half_a_day"
27791,"When my wife came home from work and we shared a steak dinner together with a bottle of wine.","food|romance|family","yes","yes","55","
USA","m","married","n","24h","at_least_one_hour"
27801,"My father bought me a bicycle.","shopping|family","no","yes","27","IND","m","single","n","24h","all_day_im_still_feeling_it"
27804,"I went to my home and given an Ice Cream Family pack to my family particularly in to the hands of my son and it creates a joyful moment."
,"family","yes","yes","46","IND","m","married","y","24h","half_a_day"
27838,"I was able to play my video game that I enjoy the most quite a bit and that made me relaxed and happy.","entertainment","yes","no","29","
USA","m","married","y","24h","at_least_one_hour"
27852,"I was able to do a full hour of research on a topic that had been nagging me.","career","yes","no","27","USA","m","single","n","24h","
at_least_one_hour"
27864,"When i got to the train station it just pulled up and i was able to get a seat.","career","no","no","30","USA","m","single","n","24h","
at_least_one_hour"
27866,"I went to support my cousin in her product launch last night and I was so happy and proud to watch her shine.","family","yes","yes","31",
"USA","f","married","n","24h","at_least_one_hour"
```

Figure 4.1: Top 20 rows in the training set of the happiness shared task dataset.

The author is in control in the moment *"Going out to a special birthday lunch for my great grandmother in law's birthday"* but not in control in *"My youngest daughter got accepted to a number of prestigious universities and accepted an offer to attend college in San Diego."*

- *Social*: Binary label describing whether or not this moment involve people (objects are not considered) other than the author. (true/false))

  Other people are involved in the moment *"Going out to a special birthday lunch for my great grandmother in law's birthday"*, or in the moment *"I received compliments on my tattoo"*, but not involved in the moment *"The weather is great today"*.

- *Concepts*: Concepts can have up to 15 possible values, and each moment can have multiple (up to 4) values separated by a piping symbol, describing the theme of the sentence. The possible concepts are as the following: *animals, career, conversation, education, entertainment, exercise, family, food, party, religion, romance, shopping, technology, vacation, weather*.

The main task of this dataset is to predict agency and social labels. Given an account of a happy moment, marked with individual's demographics, recollection time and relevant labels, what we need to do is to predict agency and social labels for happy moments in the test set, based on the small labelled and large unlabelled training data. Unfortunately, because of deep learning model is supervised, so the unlabelled training data is not used in our experiments.

The unlabelled training set contains the remaining single-sentence happy moments. The format of the unlabelled data is similar to the labelled data, without the *concepts, agency, social* labels only. The test set contains unlabelled, single-sentence happy moments, freshly collected in the same manner as the original HappyDB data. Its format is the same as the format of the unlabelled training data.

4.4 Data Preprocessing

The distribution of the social and agent labels in training data is shown in table 4.1. We can see that whereas the data are almost evenly distributed for the attribute *Social*, for *Agency*, the positive data accounts for a high proportion. This imbalance could cause the performance of our models on *Agency* to not be as good as the performance on the *Social* label.

Table 4.1: Distribution in Training Data

Label		*Agency*		Sum
		yes	no	
Social	yes	3554	2071	5625
	no	4242	693	4935
Sum		7796	2764	10560

We performed two main steps for the happy moments processing:

- **Split the sentences into word lists and omit all punctuation marks.** Sentences are processed one by one into arrays of words. All punctuation marks, including comma, period, exclamation mark, question mark, and so on, are discarded. A special case is that all abbreviations, like *I'm*, and *we're*, remain unchanged.
- **Transform the sentences into sequences and pad them to become of the same length.** Because a mathematical model can only deal with numbers, the second step is to turn sentences into numbers. First, we number all the words in sequence, starting from 1. (Index 0 is reserved for padding and unknown characters.) Then, we replace all words with their indexes, and we pad each sentence at the beginning, as needed, to make them have the same lengths. In this shared task, combining the training and the test data, there are 12,705 unique words in total. The length of the sentences ranges from 1 to 140, with an average length of 14 and a median length of 12. Both the longest and the shortest sentences appear in the test data, with hmid 1539 and 4861, respectively. After padding (or cutting), all sentences have the same length of 29, which is no shorter than the actual length for 95% of the original sentences.

We added one step for label processing:

- **Categorise *Agency*, *Social* and *Concepts*.** The class *Agency* and *Social* both contain only binary values: **yes** and **no**, which can be easily transformed into 1 and 0, whereas the class *Concepts* has 15 different values and many more combinations of them. We use the one-hot encoding method to represent all 15 concepts. Each value will be transformed into a 15-dimension array, in which the locations of the concepts that are present are marked as 1, and the others are set to 0.

4.5 Models

From bottom to top, our model comprises the input layer, the embedding layer, the convolutional layer, the dropout and pooling layer, and three detached dense layer heaps. We have compared the results of several deep learning models, like Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM). CNN proved to achieve better results in our experiments, so we will only present the results of CNN-based models in section 4.6. Among all structures we tested, the model with multi-task learning mechanism, auxiliary task *concepts*, pre-trained and trainable word embeddings, achieved the best results in the experiment.

4.5.1 Embedding Layer

In this experiment, we tried two different embedding methods: GloVe[Pennington et al., 2014] and ELMo[Peters et al., 2018]. The input requirements of these two methods are a little bit different, and we will describe them separately in the following.

GloVe: Global Vectors for Word Representation

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. The pre-trained embeddings we used are trained on 2 billion tweets corpus, with 27 billion tokens and 1.2 million vocabularies.

The embedding layer is fed with 1-D moment descriptions, which are then embedded into 2-D matrices. The size of the second dimension is 100, which means every word will be transformed into a 100-dimensional vector. For example, for a sentence with a length of 20, we first add nine zeros in the front to reach the length 29. After passing the embedding layer, the size of the output matrix will be (29×100).

The values of the embedding layer are initialized from the **glove.twitter.27B**[2] corpus file. The choice of this pretrained embedding file is due to the corpus of the dataset is also extracted from the social media texts. Then, there are two ways to handle the values during training: keep them fixed, or allow them to be updated. In the experiments, we compared the model performance between using GloVe with trainable variables, GloVe with fixed variables, and randomly initialized embedding values. We found that using pre-trained GloVe embeddings with trainable variables can lead to the best performance.

ELMo: Embeddings from Language Models

ELMo is a deep contextualized word representation that models both complex characteristics of word use (e.g., syntax and semantics), and how these uses vary across linguistic contexts (i.e., to model polysemy).

[2]https://nlp.stanford.edu/projects/glove/

We used a pre-trained ELMo embedding model, which is implemented in TensorFlow-hub [3]. About the details of usage in ELMo embedding, please see the appendix .3.

Besides the embedding layer, the model structure of ELMo is similar to that of GloVe. The input layer has a shape of $(1,)$ because ELMo can accept original sentences as input instead of tokens. Another difference is that to make the labels converge at the same epochs, the loss_weights of three labels are different from GloVe and *concepts* has a higher weight because it is the hardest one to fit.

As described in .3, ELMo embedding has two different output shapes. One is $[batch\_size, 1024]$, which is the mean of all word embeddings in the sentence; another is $[batch\_size, max\_length, 1024]$, which means all sentences all mapped into a dimension of $max\_token\_length \times 1024$.

4.5.2 Convolutional Layer

Whereas a 2-D convolutional layer suits image processing most, here we used a 1-D convolutional layer, which is often used in natural language processing [Kim, 2014]. The kernel also called a filter, has the same length as the input words. After sliding along the input sentence, each filter will generate a 1-D vector. So the size of the final output will only be related to the number of filters and length of a sentence, but not with the dimension of the word embeddings.

After the convolution, the output is fed into a dropout layer, and then the max pooling operation is performed.

4.5.3 Hard Parameter Sharing for Multi-Task Learning

Hard parameter sharing [Caruana, 1993a] is the most commonly used approach to multi-task learning in neural networks. It is applied by sharing the hidden layers between all the tasks while isolating several task-specific output layers.

Normally, after the convolutional layers, the model will be followed by a fully-connected layer and has one output (maybe with multiple dimensions) at the end. But for hard parameter sharing, the first part of the model is shared between the multiple tasks, while the layers after convolution are task-specific. Here, the class *Concepts* is treated as the third classification task, besides the two classes *Agency* and *Social*. Fig. 4.2 is the high-level description of our MTL model for three tasks. The idea is that sharing layers between the tasks could reduce the risk of overfitting for each task. Intuitively, the more tasks we are training simultaneously, the more our model will try to represent all of the tasks, leading to a lower chance of overfitting on a single task. Our proposed MTL model still achieves good results, as shown in the next section.

[3] https://tfhub.dev/google/elmo/3

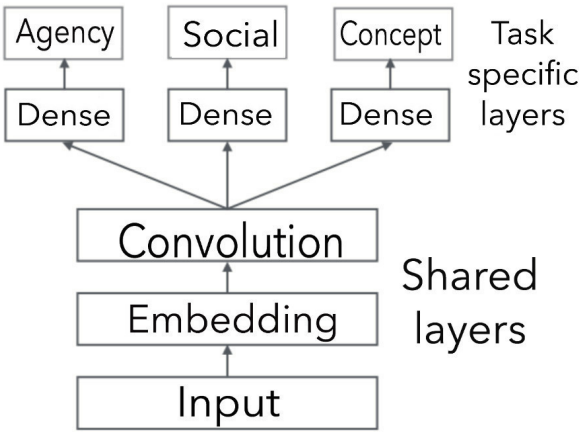


Figure 4.2: An example of hard parameter sharing for the three tasks.

4.6 Experimental Results and Evaluation

During training, we use mini-batch gradient descent with batch size 512 and the Adam optimizer [Kingma and Ba, 2014] is used with a learning rate of 0.001. In order to get stable and robust test results, we use k-fold cross-validation (see section .2.1 in the Appendix for details) with k value 5[4]. For the system run submissions in the shared task, we use 80% of data for training and 20% as validation, and we report the results on the test data from the shared task.

4.6.1 Comparison between STL and MTL Results

Table 4.2 shows the results of several models [5]. From left to right, the abbreviations of the models refer to:

CNN: Convolutional Neural Network model, with randomly initialized embedding layer. As this is a single-task model, two models are trained separately for the two labels *Agency* and *Social*.

CNN+MTL: Convolutional Neural Network model with randomly initialized embedding layer. It is a multi-task model, which contains shared layers and separate layers from two labels.

CNN+MTL+GloVe, fixed: Convolutional Neural Network model with embedding layer which is initialized from pre-trained GloVe embeddings. The values of word embeddings are not allowed to be updated during training. It is also a multi-task model and contains shared layers and separate layers.

CNN+MTL+GloVe, trainable: Convolutional Neural Network model with embedding layer which is initialized from pre-trained GloVe embeddings. The values of word embeddings are updated during training. Except for the trainability of the embedding layer, all other configurations in this model are all the same with the previous one.

Table 4.2: Comparison between STL and MTL results.

Model	CNN		CNN+MTL		CNN+MTL+GloVe, fixed		**CNN+MTL+GloVe,trainable**	
Label	Agency	Social	Agency	Social	Agency	Social	**Agency**	**Social**
Accuracy	0.712	0.770	0.804	0.858	0.832	0.893	**0.839**	**0.902**
Precision	0.664	0.799	0.759	0.860	0.815	0.894	**0.802**	**0.902**
Recall	0.689	0.778	0.744	0.856	0.748	0.893	**0.811**	**0.903**
F1-score	0.670	0.767	0.751	0.857	0.761	0.892	**0.798**	**0.901**

In table 4.2, we can see that the results of models are increasingly better from left to the right. Let us look into them pair by pair.

[4]We were not able to not use k=10 because training the deep learning took too long. Most research in deep learning uses k=5 or does not do cross-validation at all, because of the impractical running times.

[5]All word embeddings are of dimension 100.

For the first two models, *CNN* and *CNN+MTL*, the application of multi-task learning improves the model performance by about ten percent. This result is as we expected. Multi-task learning can encourage labels to use information from other tasks and prevent the model from overfitting on each label. Because the embedding layer is initialized with random values, the model is more likely to be overfitted on the tasks during the training without the guidance from word embeddings.

For the second and third models, *CNN+MTL* and *CNN+MTL+GloVe,fixed*, it clearly shows how GloVe embeddings help improve the model's performance. We note that in the second model, *CNN+MTL*, the values in the embedding layer (which are randomly initialized) are allowed to be updated during training, to make them fit the task. Despite this, the performance of *CNN+MTL* is still worse than that of *CNN+MTL+GloVe*, which does not update the word embeddings during training. A possible explanation is that pre-trained word embeddings have a strong power of guidance for the model, to represent the meaning of the texts and prevent the embeddings from overfitting on the training data. The size of the training corpus is much lower than that size of the corpus used for building the pre-trained embeddings.

The two models in the last pair, *CNN+MTL+GloVe, fixed* and *CNN+MTL+GloVe, trainable*, show us the difference in performance between trainable embeddings and fixed embeddings. As we said above, pre-trained word embeddings can use the power from the large external corpus to guide the small internal training data. But this does not mean the expressions from the external corpus will be perfect for the task-specific dataset. A few updates in the training step can make them fit together better. We think that the adjustment must be delicate and modest; a large learning rate can easily destroy the expressivity of the original embeddings.

The loss, precision, recall and F1-score graph of the model *CNN+MTL+GloVe, trainable* during the last fold of training is shown in Fig. 4.3. We stopped at epochs 8 because the two main labels, *Agency* and *Social* have been fitted at there. The model will be overfitted if continuing training on the auxiliary label *Concepts*, which is not what we want. We tried to increase the weight of label *Concepts*, but its fitting is always later than the other two, so we have to focus on the main labels.

All the results in table 4.2 are from models with an embedding layer of dimension 100. We have tested the models on other dimensions, like 50 or 200, but the results did not change much.

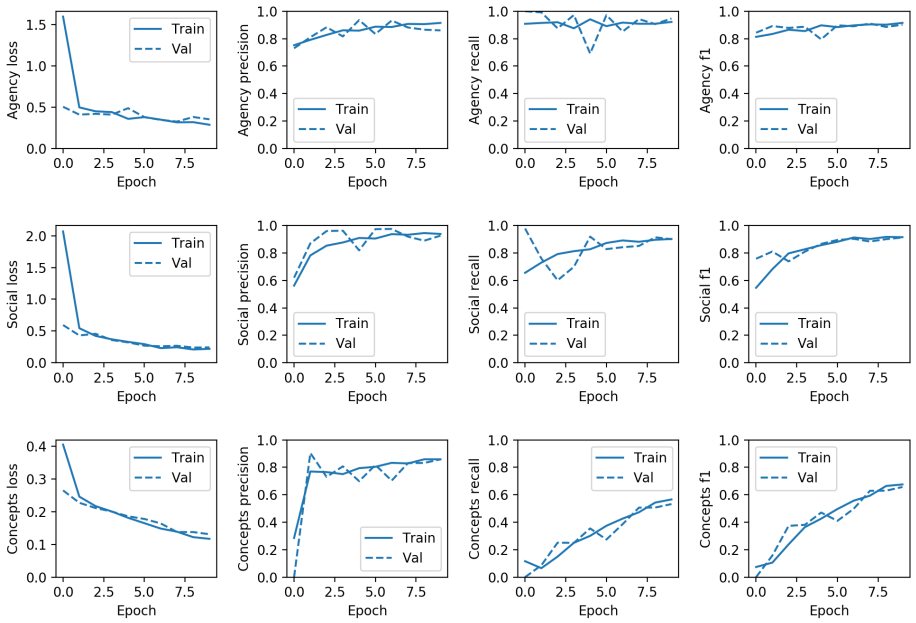


Figure 4.3: Loss,precision,recall and F1-score graph of model *CNN+MTL+GloVe, trainable*.

4.6.2 Experiments with and without the Auxiliary Task *Concepts*

As we mentioned before, the shared task has only two tasks, *agency* and *social*. The label *concepts* is added by ourselves as an auxiliary task for supporting the multi-task training. In table 4.2, all the models whose name contain "MTL" are trained on the three labels, instead of the original two. Then a question appears here: does the auxiliary label *concepts* actually help the training in multi-task learning?

We trained two more models, based on the structure of model *CNN+MTL+GloVe, fixed* and *CNN+MTL+GloVe, trainable*, but this time the MTL only contains the two original labels, *agency* and *social*. The results are shown in table 4.3.

We can see that when changing the number of tasks from three to two, the result of label *Agency* does not change much, whereas the result of the label *Social* decreases with about 2 percents, both in the fixed and trainable embeddings models. Two percent is not a large drop, but it still shows that the auxiliary task helps in the multi-task learning. We also found that the auxiliary task makes models more stable, which we could see during the cross-validation experiments.

Table 4.3: Performance with and without auxiliary task *concepts*.

Model	MTL(3 labels), fixed		**MTL(2 labels), fixed**		MTL(3 labels),trainable		**MTL(2 labels), trainable**	
Label	Agency	Social	Agency	Social	Agency	Social	Agency	Social
Accuracy	0.832	0.893	0.832	0.878	0.839	0.902	0.848	0.872
Precision	0.815	0.894	0.813	0.884	0.802	0.902	0.819	0.884
Recall	0.748	0.893	0.761	0.880	0.811	0.903	0.781	0.875
F1-score	0.761	0.892	0.766	0.877	0.798	0.901	0.791	0.871

4.6.3 Comparison between GloVe Embedding and ELMo Embedding Results

We have already achieved reasonable results by the combination of GloVe embedding and multi-task learning, but we feel that in the current structure, GloVe has reached the upper bound of its ability. So we turn our attention to another popular word embeddings method, ELMo. (BERT has not been popular at that time.)

The usage of ELMo is totally different from GloVe. We will omit the details here, but they are described in section .3. Generally speaking, given a pre-trained GloVe embedding model, all word embeddings are fixed in the corpus, so the deep learning model[6] can easily load them into the built-in embedding layer using a map function, without any other changes. However, when it comes to ELMo, the embeddings are extracted from the bi-LSTM network and changed during the training, which means that the ELMo embedding must be treated as a separate module that is not natively supported by the embedding layer.

ELMo supports two different embedding outputs: one with the shape of $[(batch\_size,)1024]$ and another $[(batch\_size,)max\_token\_length, 1024]$. We compared the results of these two shapes with the result from the GloVe embedding, as shown in Table 4.4.

Table 4.4: Comparison between GloVe embedding and ELMo embedding.

Model	CNN+MTL+GloVe,trainable		MTL+ELMo(1-D output), trainable		CNN+MTL+ELMo(2-D output), trainable	
Label	Agency	Social	Agency	Social	Agency	Social
Accuracy	0.839	0.902	0.831	0.899	0.850	0.906
Precision	0.802	0.902	0.798	0.900	0.808	0.905
Recall	0.811	0.903	0.766	0.899	0.796	0.906
F1-score	0.798	0.901	0.771	0.898	0.802	0.905

The result from ELMo of the shape $[(batch\_size,)1024]$ is worse than the best configuration of GloVe. It is logical to us if considering the model structures: when the output shape of ELMo is 1-D, the convolutional layer cannot be used after the embedding layer, so a model with only feedforward layers will have worse performance than a model with convolutional layers. Whereas for the ELMo embedding with shape $[(batch\_size,)max\_token\_length, 1024]$, the performance of the model is very close to the best GloVe model. However, considering that the implementation effort is much more higher and the training time of ELMo is five times that of the GloVe model, we do not recommend to use it as the primary embedding method again in the future, considering the low improvement it brings. Especially after the coming of BERT[Devlin et al., 2018], which is a more powerful word embedding model.

[6]We used Keras and TensorFlow.

4.6.4 Parameters Used in the Submitted System Run

In the workshop, we submitted ten system runs for evaluation. We trained models on the training data (the split ratio we used was 80%:20%, which means 80% of the data was used for training and 20% for validation). Then we applied our models on the provided unlabelled test data. A system description paper [Xin and Inkpen, 2019] was published in the workshop's proceedings.

We selected our best model (CNN with multi-task learning and trainable GloVe embeddings used for initialization. Unfortunately, model with ELMo had not been finished at that time), with small variations. Generally, the parameters used during the ten runs were all the same except the dimension of the embedding layer and the number of training epochs. During the preprocessing of the test data, all sentences were padded to become of length 29, the same as we did for the training data. In the neural network architecture, there were 64 filters in the 1-D convolutional layer with a kernel size of 5. The dropout layer was connected, and the dropout rate was set to 0.2. For the multi-task learning part, the three tasks had equal weights.

The different dimensions of the embedding layer and the number of epochs for each run are shown in table 4.5.

Table 4.5: Parameters for the Submitted System Runs

No.	**File Name**	**Emb. Dim.**	**Epochs**
1	20181205_174718_CNN_MTL_Embedding_100D_10epochs	100	10
2	20181205_175239_CNN_MTL_Embedding_50D_10epochs	50	10
3	20181205_175746_CNN_MTL_Embedding_200D_10epochs	200	10
4	20181205_180940_CNN_MTL_Embedding_25D_10epochs	25	10
5	20181205_181415_CNN_MTL_Embedding_25D_50epochs	25	50
6	20181205_181703_CNN_MTL_Embedding_50D_50epochs	50	50
7	20181205_182149_CNN_MTL_Embedding_100D_50epochs	100	50
8	20181205_182951_CNN_MTL_Embedding_200D_50epochs	200	50
9	20181205_184403_CNN_MTL_Embedding_200D_100epochs	200	100
10	20181205_194544_CNN_MTL_Embedding_50D_200epochs	50	200

4.7 Comparison to Related Work

There were eleven teams that participated in the shared task. Fig. 4.4 shows the difference between the result of our team and those of the other teams.

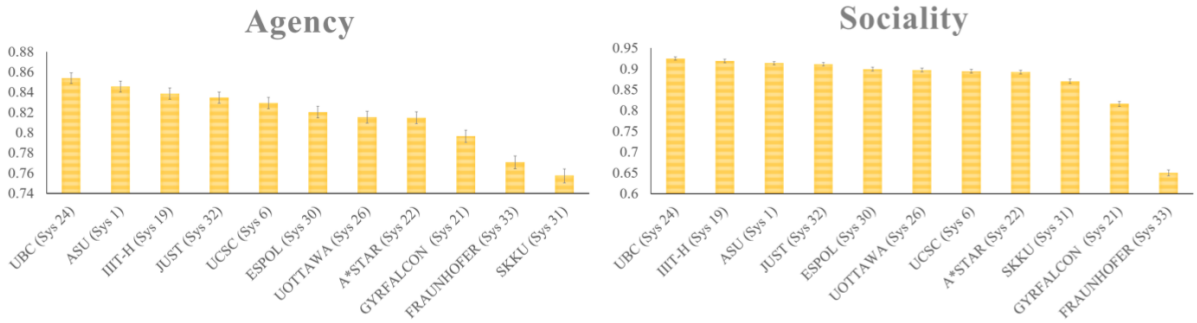


Figure 4.4: Accuracy scores for the best performing system runs on CL-Aff Task 1 for each of the participating teams.

Our result (from the model *CNN+MTL+GloVe,trainable*) is the seventh for the label *Agency* and the sixth for the the label *Social*. The ranking is not as good as we wanted, but the top eight results are very close. Unfortunately, at that time we only implemented the model with the GloVe embeddings. Our models with the ELMo embedding 4.6.3 were finished after the workshop. According to [Jaidka et al., 2019], most of the teams that obtained the top scores used ELMo embeddings in their experiments.

Following is an overview of the models used by the other teams who participated in the shared task [Jaidka et al., 2019].

Arizona State University (ASU)

The team from ASU proposed a Word Pair Convolutional Model (WoPCoM) to accomplish Task 1 [Saxon et al., 2019]. The proposed model is motivated by the hypothesis that a small set of word-pair features are important for capturing the agency/social nature of happy moments. They trained a convolutional neural network (CNN) to predict on the unlabelled data.

University of California Santa Cruz (UCSC)

The UCSC team participated in both tasks [J. et al., 2019]. For Task 1, they explored the use of syntactic, emotional, and survey features with semi-supervised learning, specifically experimenting with XGBoosted Forest and CNN models. For Task 2, the team trained similar models to predict concepts, and based on the difficulty of doing this, they hypothesized about the nature of the themes in the happy moments.

International Institute of Information Technology Hyderabad (IIIT-H)

The IIIT-H team employed an inductive transfer learning technique (ITL) [Syed et al., 2019]. They pre-trained a AWD-LSTM neural net on the WikiText-103 corpus, and then introduced an extra step to adapt the model to the happy moments data.

Gyrfalcon

The team from Gyrfalcon Technology, California, proposed an algorithm to map English words into squared glyphs images [Sun et al., 2019]. Then, they applied a 2-D-CNN model over these images in order to capture the expressed sentiment.

A * STAR

The IHPC-A*STAR team participated in both tasks [Gupta et al., 2019]. For Task 1, they used the emotion intensity from the happy moments to predict the Agency and Social labels. They defined a set of five emotions (violence, joy, anger, fear, sadness) and used a previously developed tool, CrystalFeed, to label each moment with the corresponding five emotion intensities. Combining these features with additional word-embedding features, they trained a logistic regression model. For Task 2, the team explored how these emotions are expressed across the different concept labels.

University of British Columbia (UBC)

The UBC team primarily experimented with different embedding methods, such as CoVe and ELMo, on deep neural networks [Rajendran et al., 2019]. They modeled their neural networks as LSTM networks (biLSTM), with and without an attention mechanism.

Escuela Superior Politecnica del Litoral (ESPOL)

The ESPOL team proposed a semi-supervised adaptation to traditional k-means clustering using neural networks [Torres and Vaca, 2019].

Sungkyunkwan team (SKKU)

The SKKU team used a semi-supervised approach [Cheong et al., 2019]. They built four one-class auto-encoder models, one for each of: social, non-social, agentic, and non-agentic moments. Each auto-encoder model had a deep learning architecture consisting of two neural networks, one for encoding the the input and the other for reconstructing the compressed vector.

Jordan University of Science and Technology (JUST)

The JUST team proposed used a Recurrent Convolutional Neural Network and combined words with their context in order to get a more precise word embedding [Talafha and Al-Ayyoub, 2019].

Fraunhofer (FKIE)

The team from Fraunhofer FKIE trained a three-layer CNN [Claeser, 2019]. They experimented with using different embeddings, including FastText and GloVe. Additionally, they experimented with splitting the dataset by the demographic location of the author and showed that training separate classifiers on the splits enhanced performance.

4.8 Error Analysis

We reviewed 507 wrongly-predicted (either *Agency* or *Social*, or both) happy moments in the first fold of our best model *CNN+MTL+GloVe, trainable* in 4.6.1, and found some interesting patterns.

4.8.1 Label *Agency*

Let us first look at label *Agency*. Among all 507 wrongly-predicted labels, *Agency* takes up 325 cases, which are composed of 199 false positive cases and 125 false negative cases.

False positives in *Agency*

Most of the false positive cases ((Expected *false* : Predicted *true*)) have a similar sentence pattern: starting with the subject "I". For example:

- *I found out that my band gets to play a huge local music festival.*
- *I got nominated for teacher of the year at school.*
- *I got my paycheck from work.*
- *I got hired as a personal trainer yesterday and it made me so happy because its what I wanted to do for a long time.*

These sentences are really hard to distinguish for the model. The subject "I", followed by a verb, seems to make the model think that an action is launched by the author, but in fact it is only a statement of fact.

We also found an ambiguous situation: should an expression of emotion be considered as in control or not in control? For example, we found that the expression of happiness is considered sometimes as *true* and sometimes as *false* for the label *Agency*:

- *I am happy to be having my yard sale next week.* (Considered as *true*)
- *I was happy when my son asked me to watch a show with him.* (Considered as *false*)

We look back to the definition of *Agency*: whether or not the author is in control under the situation, i.e., whether or not objective facts are transferred by the person's subjective will. The emotion is derived from the inside of a person and is controlable and changeable. We believe it should be always treated as *true Agency*. But unfortunately, the annotation rules used when producing the dataset did not mention how to treat emotions and probably the annotators used their own interpretations in these cases.

False negatives in *Agency*

For false negatives (Expected *true* : Predicted *false*) in *Agency*, the major pattern is that the subject is someone else. For example:

- *My friend and their family came to stay with me on vacation.*
- *My friend tells me I'm funny.*
- *The person taking my order at the drive-thru was very friendly.*
- *The girl I wanted to date sent me a message me out of the blue.*
- *My daughter told me that I was the best mommy in the world.*

The actions are initialized by someone else, the author is also involved. Over two thirds of all wrongly-predicted false negatives follow this pattern.

There are also some ambiguous false negatives:

- *My family got a new puppy.* ("Family" is ambiguous here)
- *my boy born with a lovely cry.* (This one should be a wrongly-labelled data.)

4.8.2 Label *Social*

For *Social*, 223 moments are wrongly predicted on this label among all the 507 wrongly-predicted for all the labels. Wrong-predicted *Social* is composed of 29 false positive cases and 194 false negative cases.

False negatives in *Social*

We cannot find a common pattern in the false positive part as the sample size is 29, which is too small compared with the false negative part. But on the other hand, the big difference between the amount of false positives and false negatives shows that maybe the model is not sensitive enough when predicting the *Social* label.

In the false negatives part (Expected *true* : Predicted *false*), passive voice appears frequently. For example:

- *I was given a bigger raise compared to the one I got last year at work.*
- *I was praised at work for completing a task quicker than expected.*
- *Today I was offered a full time babysitting job.*
- *I was also told that I was pretty.*

The reason is easy to image. In passive voice, someone else is implicitly involved although not explicitly mentioned, but our model cannot detect this situations.

Wrongly-labelled data in *Social*

Even though we already found some wrongly labelled data for *Agency*, the issue for *Social* seems to be worse. Here are some examples:

- *I had a really delicious dinner last night.* (labelled as *true*)
- *I had fun on a trip to Las Vegas.* (labelled as *true*)
- *I was hungry, and was surprised with breakfast.* (labelled as *true*)
- *When my hit amount hit $91.00 mark today for this week I was extremely happy.* (labelled as *true*)
- *I participated in a physical therapy session.* (labelled as *true*)
- *I went to see a movie with my friends at the theater.* (labelled as *false*)
- *I was very happy when I enjoyed a Chipotle dinner outside with my husband on a nice sunny day.* (labelled as *false*)
- *I spoke to someone about having an engagement ring made.* (labelled as *false*)

Cat and Dog

Another big issue for the label *Social* is how to treat pets. Unfortunately, this was not mentioned in the annotation rules either, so all the labels are based on the interpretation of annotators, which seems to have used different polarities:

- *I recently got a new cat.* (labelled as *true*)
- *My cat snuggled with me on my bed earlier.* (labelled as *true*)
- *My dog behaved when confronted by another dog on our daily walk.* (labelled as *true*)
- *I played with a dog in the park.* (labelled as *true*)
- *I adopted a great cat from the local shelter.* (labelled as *false*)
- *One of my cats meowed at me before I left my house today as if to say: 'I'll miss you while you're gone.'* (labelled as *false*)
- *My dog kissed my face this morning.* (labelled as *false*)

4.9 Summary and Directions of Future Work

In this chapter, we present our multi-task deep learning model for detecting happiness ingredients using two different embedding methods, GloVe and ELMo. Our experiments show that the model works well on the provided happiness text data. We obtain acceptable accuracy and F1-score, especially on the label *Social*.

Based on the experiments, we have the following conclusions:

- First, multi-task learning models have better results than single-task learning models;
- Second, pre-trained word embeddings can help improve the model performance;
- Third, allowing the word embeddings update themselves during training can get better results than using fixed word embeddings;
- Forth, the auxiliary task helps in the multi-task learning, even though the help may be tiny on some labels;
- Last but not least, the ELMo embedding has a very close but slightly better result over the GloVe embedding, whereas the implementation of ELMo is much more complex.

Although our best model achieves good results, there are still some methods we can use to improve its performance. One direction of future work is to also learn from the unlabelled data (70,000 instances). We did not use the unlabelled in our current model due to time constraints. We propose to use a bootstrap algorithm: to run our current best model on the unlabelled data, then add to the labelled training data the best of the automatically-labelled instances, namely the ones for which the confidence in the prediction is high for both classes (Social and Agency); then to retrain our model on the enhanced training data. The model trained by this bootstrapping method might work better, but only if we do not add too much noise to the training data.

Another direction of future work is to make use of other information provided in the training data, such as age, gender, location, marital status and parental status.

We experiment with detecting concepts as a separate task while training the MTL model, but we could further apply the model to predict concepts on the test data. Then we can use these automatically-detected concepts when we run the model on the test data in order to obtain results for the multi-task model with three tasks.

Chapter 5

Experiment 2: Multi-Task Deep Learning for Detecting Disclosure and Support

5.1 Introduction

There is a growing interest in understanding how humans initiate and hold conversations. The effective understanding of conversations focuses on the problem of how speakers use emotions to react to a situation and to each other.

The CL-Aff Shared Task 2020: *Get it #OffMyChest* [1] was proposed based on this motivation. The workshop introduced the OffMyChest conversation dataset and invited submissions for the Computational Linguistics Affect Understanding (CL-Aff) Shared Task on modelling interactive affective responses. The shared task was part of the AffCon workshop 2020 @ the AAAI 2020 conference.

We participated in this shared task because it has interesting labels, and the task is related to our thesis topic, multi-task learning. We submitted two system runs and a model description paper [Xin and Inkpen, 2020] and obtained some of the best results among all the participants.

5.2 Task Description

There are two sub-tasks in this shared task. The data consists of **English** sentences sampled out from casual and confessional conversations among Redditors from the *#CasualConversations* and the *#OffMyChest* forums, labelled with their informational, social, and affective characteristics.

[1] https://sites.google.com/view/affcon2020/cl-aff-shared-task

Task 1 The first task is a semi-supervised learning task: Predict labels for Disclosure and Supportiveness for sentences based on a small set of labelled training data and large set of unlabelled data.

Task 2 The second task is an unsupervised task: Propose new characterizations and insights to model conversation dynamics.

As task 2 is an open-ended problem and does not have a strong relationship with our thesis topic, we focused only on task 1 in our experiments.

The OffMyChest conversation dataset is provided for this shared task. It included twelve thousand samples. Each entry contains a sentence and six binary labels, namely *Information_disclosure, Emotional_disclosure, Support, General_support, Info_support*, and *Emo_support*. The definition of each label is as follows:

- **Information_disclosure** Comment contains personal information about the author or about people that the author mentioned, such as age, what their job is, where their lives, interests, diseases, physical appearance, or behaviour.

  Examples: ("I'm now 65 years old."; "I've worked with kids with ODD and autism."; "I live in West Philly."; "Sounds like our bipolar kid."; "She posted a screenshot of his porn history (gross)"; "My mum told me that she was sexually abused as a kid."

- **Emotional_disclosure** This comment contains the author's positive or negative feelings, such as appreciation, amusement, concern, frustration, excitement, happiness, being overwhelmed, humiliation, anger, sorry, agony, anxiety, depression, fears, or pain.

  Examples: "My only concern was for my son."; "Fuck me that is beautiful."; "Thanks for sharing the story."; "My heart melted reading this xx"; "I'm not crying, you're crying!"; "I'm literally too jealous"; "My heart is breaking for you."

- **Support** This comment is offering support to someone, either through sympathy, encouragement, or advice.

  Examples: "Good luck, this shit is tough"; "Good luck! but I'm afraid I have no advice"; "Hey, you tried your best"; "Have you tried family therapy?"

- **General_support** The comment is offering general support through quotes and catchphrases.

  Examples: "What's the worst that could happen?"; "You only die once."; "All's well that ends well."

- **Info_support** The comment is offering specific information, practical advice, or suggesting a course of action.

  Examples: "I wouldnt.."; "You shouldn't.."; "You can't..".; "Why didn't you try this?"; "Please talk to a professional."

- **Emo_support** The comment is offering sympathy, caring or encouragement.

  Examples: "Good luck, this shit is tough"; "Good luck! but I'm afraid I have no advice"; "You sound like a great person"; "I'm so sorry."; "That's a great story."

5.3 Data Description

The dataset from this shared task is split into three parts:

- **Unlabelled training set:**

  Unlabelled POSTS: The top posts in 2018 in *CasualConversations* and *OffMyChest* mentioning any of the keyword terms. Posts that are parents of comments in the training and test sets are separately identified.

  Unlabelled COMMENTS: Over 420,000 sentences extracted from 130k comments posted to "POSTS".

- **Labelled training set:** In total 12,860 labelled sentences, extracted from the top comments posted to "POSTS". The major labels have two categories: ***Disclosure*** and ***Supportiveness***.

  Disclosure is further categorized into informational or emotional disclosure, while *Supportiveness* is further categorized into general, informational and emotional supportiveness.

- **Test set:** About 3,000 unlabelled sentences, extracted from the top comment and posted to "POSTS".

Because our method does not involve unsupervised learning and the labels of the test set were not released by the time we ran our experiments, we will focus on the **labelled training set** for most of the experiments in this chapter.

There are 12,860 entries in the **labelled training set**. Each of them is composed of 16 labels, including: *sentenceid, author, nchar, created_utc, score, subreddit, label, full_text, wordcount, id, Emotional_disclosure, Information_disclosure, Support, General_support, Info_support, Emo_support*. Among these 16 labels, what we need to pay attention to are *Emotional_disclosure, Information_disclosure, Support, General_support, Info_support,* and *Emo_support*, as they are our targets to predict. Fig. 5.1 shows the top 20 rows in the dataset.

The distribution of the six labels in training data is shown in table 5.1. We can see that in all the labels, the negative data accounts for a high proportion, especially for the label *General_support*, in which the negative data reach a proportion of 94.7%. This tells us to pay attention to the class weights during training. Nonetheless, it is likely that the result on the label General_support will be among the lowest since it has the highest class imbalance.

Table 5.2 shows the token analysis of the training data. As shown in the table, although the maximum token length reaches 171, 95% percent of sentences have a length of no more than 34. So the target length in sentence pre-processing is chosen to be 34 (longer sentences will be truncated). After retrieving the word embedding vectors, all sentences will be transformed into vectors with shape ($34 \times embedding\_dimension$).

```
sentenceid,author,nchar,created_utc,score,subreddit,label,full_text,wordcount,id,Emotional_disclosure,Information_disclosure,Support,
General_support,Info_support,Emo_support
36,LeniaLilac,135,1532518450,81,offmychest,husband boyfriend,Get two nice notebooks and write it down for each of them.,12,91px39,0,0,0,0,0,0
47,call-me-mama-t,115,1533351628,59,offmychest,husband boyfriend,I<U+0092>m sobbing reading this thread!,5,91px39,1,1,0,0,0,0
57,BluebirdOnAHill,46,1532513532,35,offmychest,husband boyfriend,Hope you have a nice day,6,91px39,0,0,1,1,0,0
58,MenudoMenudo,157,1532519594,34,offmychest,husband boyfriend,"My wife came in when I was around half way through this and asked why I was all
choked up and watery eyed, so we read it together and now we're both crying.",33,91px39,1,1,0,0,0,0
74,modistabeat,191,1533722813,30,offmychest,husband boyfriend,I am crying a lot of happy tears right now.,10,91px39,1,0,0,0,0,0
82,throwmygaysexlife,52,1534129785,24,offmychest,husband boyfriend,Some asshole is cutting onions around me right now.,9,91px39,1,1,0,0,0,0
150,BSBees,579,1533298109,52,offmychest,husband,"He<U+0092>s my father in every sense of the word but name, I still call him by his first name
but only because we are both used to it and he doesn<U+0092>t mind a bit.",34,946qw9,1,0,0,0,0,0
153,BSBees,579,1533298109,52,offmychest,husband,Stepdad will be the one walking me down the aisle when I get married.,14,946qw9,1,1,0,0,0,0
163,Arogani,561,1533305321,30,offmychest,husband,"While we were moving everything into our new house, I was trying to get his attention.",16,
946qw9,0,1,0,0,0,0
170,jcooli09,934,1533297995,22,offmychest,husband,"Having a step parent can be pretty awkward for a kid, the whole process of divorce and
remarriage can leave them not really knowing what their relationship is supposed to look like.",32,946qw9,1,1,0,0,0,0
180,#NAME?,696,1533277592,21,offmychest,husband,"There was a lot of stress, insecurity and fear in my life on top of going through puberty and
I never really bonded with my step mum because if it.",30,946qw9,1,1,0,0,0,0
213,r_o_k,213,1533289332,6,offmychest,husband,"It<U+0092>s likely she was calling you her stepdad to other people, as I was, ages before she
said it around you.",21,946qw9,0,1,0,0,0,0
215,prettyunicornpeni,304,1533347852,6,offmychest,husband,"Aw man, this is cute!",5,946qw9,1,0,0,0,0,0
259,missyanntx,464,1533309731,3,offmychest,husband,That's wonderful. :) My step-dad has been around for 30 yrs now.,12,946qw9,1,1,1,1,0,1
326,Forgottenbirthdays,497,1537488905,36,offmychest,husband wife,"Her husband could still be alive, from what we understand - and my mom's
siblings have the 50p  that she wrote for him ready and waiting.",27,9hgf7h,0,1,0,0,0,0
328,MindyS1719,345,1537470464,32,offmychest,husband wife,She called me 5 times at work.,7,9hgf7h,1,1,0,0,0,0
451,arthas_LK,770,1538168314,54,offmychest,husband boyfriend,You know it seems like what Ford said was true ... ,10,9joq0o,0,1,0,0,0,0
475,Dakar-A,505,1538168568,40,offmychest,husband boyfriend,This whataboutism is not helpful to the cause you purport to be behind and instead
serves to actively diminish the public view of the severity of it when it is brought as an unwarranted response to a woman who was raped or
sexually assaulted.,44,9joq0o,1,0,0,0,0,0
484,PM_me_kitten_photos,229,1538167861,27,offmychest,husband boyfriend,"I can't imagine someone bragging to friends about it, at least with the
vast majority of people I know.",19,9joq0o,1,1,0,0,0,0
```

Figure 5.1: An overview of the top 20 rows in the CL-Aff OffMyChest dataset. The first line is the header of the dataset.

Table 5.1: Data Distribution in the Training Set

Label	True	False
Emotional_disclosure	3948	8912
Information_disclosure	4891	7969
Support	3226	9634
General_support	680	12180
Info_support	1250	11610
Emo_support	1006	11854

Table 5.2: Word Pre-processing Results

	Value
Max token length	171
Min token length	1
Mean token length	15.07
Median token length	13
Number of unique tokens	11,460
Total number of tokens	193,837
Length that covers 95% of sentences	34

5.4 Word Embeddings

The pre-processing steps included standard operations like splitting the sentences into words, omitting all punctuation marks, transforming the sentences into sequences and padding them to the same length (34 in this case). The steps have been described in detail

in section 4.4 of experiment 1, so we will omit them here.

The word embedding method we chose is BERT [Devlin et al., 2018]. Unfortunately, as the GPU we had available did not have sufficient memory, we could not use the BERT embedding as a layer in the model. Instead, we use bert-as-service [Xiao, 2018] to do word embeddings beforehand. By losing some performance in value updating, this allowed us to use less memory and reduced the computation time. We used the default configuration and the bert-as-service configuration called *"ELMo-like contextual word embedding"*. The former one aims to generate sentence embeddings, and the latter one will create embeddings that have a similar shape to the ELMo embeddings [Peters et al., 2018]. In other words, *"ELMo-like contextual word embedding"* configuration will generate separated embedding for all words in the padded sentences.

In fact, before using BERT, we actually tried the ELMo embedding , but it was very time consuming and its result was worse than bert-as-service. Therefore, in this experiment, we switched to BERT.

Finally, there are two word embedding files we obtained from BERT, one with shape $12,860 \times 1,024$, and the second with shape $12,860 \times 34 \times 1,024$. $12,860$ is the number of instances in the training set, while 34 is the objective length we defined for each sentence, and $1,024$ is the dimension of each word (or the whole sentence in the default configuration, namely sentence embedding).

5.5 Models

In experiment 1we showed the power of multi-task learning compared with single-task learning. Then in experiment 2, we want to fully utilize the power of multi-tasking learning with hard parameter sharing [Ruder, 2017] on different structures of neural networks.

Multi-task learning can help the model training by sharing the information from one label to the other. In terms of neural networks, parameter values in neurons are updated by gradient descent from all labels, which prevents the model from overfitting on a specific task. Furthermore, MTL can reduce the impact of noise patterns. In general, when training a model on a task using noisy datasets, we need to ignore the data-dependent noise and to learn good patterns based on other features. Because different tasks have different noise patterns, a model trained for multiple tasks can generate a more general representation and can average the noise patterns on the different tasks [Ruder, 2017].

In this task, we tried several models. We will describe the model structures and compare their results in the following section.

5.5.1 Basic MTL Model

The first and basic model has a similar structure with the model in experiment 1. As shown in Fig. 5.2, from bottom to the top, there is a shared layer part and task-specific part. The shared layer part contains the *Embedding input layer*, while the task-specific

part contains the *fully-connected dense layer* and the *Output layer*. The structure is quite simple and uses only the multi-task learning mechanism. The activation function used for all layers except the output layer is leaky-Relu, while in the output layers, it is the sigmoid function. This model will serve as a baseline for other models.

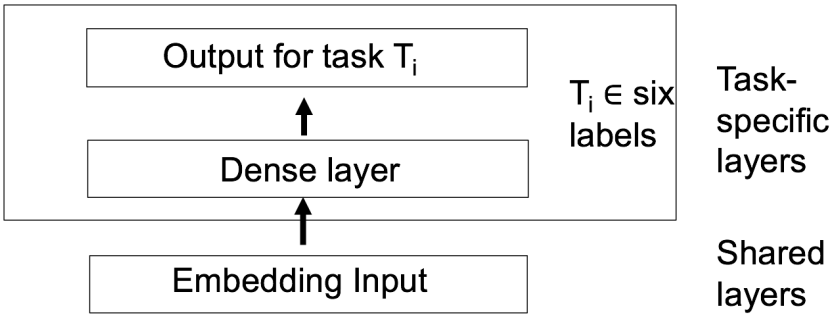


Figure 5.2: Basic Multi-task learning model

5.5.2 Intermediate MTL Model with Conv1D and PRelu

As we can see from Fig. 5.2, the basic model looks neither powerful nor "deep" for a deep learning network, so we added some ingredients into the model to help it deal better with natural language processing problems. As shown in Fig. 5.3, in the shared layer part, two more layers have been added depending on the dimension of the input embedding files. For the 2-D input (shape $12,860 \times 1,024$), two fully-connected dense layers are added, while for the 3-D input (shape $12,860 \times 34 \times 1,024$), one 1-D convolutional layer, one pooling layer and one fully-connected dense layer are added. Furthermore, another fully-connected dense layer is added into the task-specific part for all tasks.

Besides the leaky-Relu activation function we used, we tried another activation function for all layers except the output layer called PRelu[He et al., 2015]. Compared with leaky-Relu, PRelu activation function provides a slightly better accuracy, at the price of a longer training time.

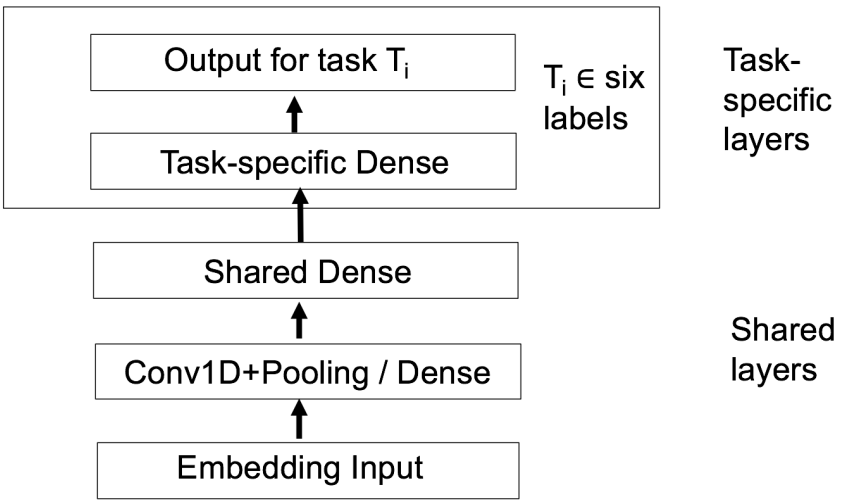


Figure 5.3: MTL model with 1-D convolutional layer and PRelu activation function

5.5.3 MTL with a Combination of Three Embedding Methods

We have discussed several different embedding methods in section 2.2.1, including *GloVe, ELMo* and *BERT*. Two of them, *GloVe, ELMo*, have been tested in experiment 1. As different embedding methods have different text representation algorithms, they may capture distinct hidden features in the corpus. Therefore, if the model can use different embeddings at the same time, it may be stronger than using a single one.

We tried different feature extracting layers that are located above the embedding input layer, like Conv1D, Bi-LSTM and Bi-GRU methods. These methods also have different abilities to refine the features from the embeddings.

The model we built contains all of the embeddings and feature extracting layers, as shown in Fig. 5.4. We want to see if it can achieve better results than using a single embedding input.

Each embedding input, BERT, ELMO and GloVe, will connect to all three types of the feature extracting layers, including Conv1D, Bi-LSTM and Bi-GRU. So in total, there will be 9 different intermediate outputs. These outputs are flattened and concatenated into a dense layer, which then connects to the task-specific layers.

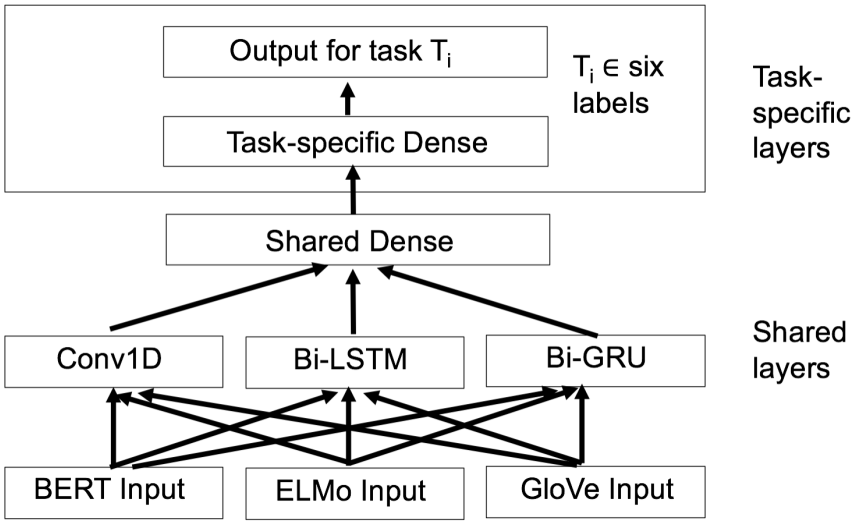


Figure 5.4: MTL with a combination of three embedding methods and three feature extracting layers

5.5.4 MTL with LIWC as Additional Input

LIWC (Linguistic Inquiry and Word Count)[Pennebaker et al., 2015] is a text analysis program. It can calculate the degrees to which various categories (groups of words) are present in a text, and has the ability to process formal and informal texts.

After pre-processing the training data through the LIWC 2015[2], a new labelled file is generated, which contains 109 columns and 12,861 rows. Excluding the 17 columns inherited from the original training file, finally, we get 92 new labels automatically annotated by LIWC 2015. As the labels generated from LIWC are real values, we needed to normalize them before entering them as input into the neural network.

Because the labels generated by LIWC 2015 are based on the statistics of the training data, it may contain helpful information for predicting the target labels we want. In the first step, we treated these labels as an additional input of the model, as shown in Fig. 5.5, to see if this can help improve the prediction result.

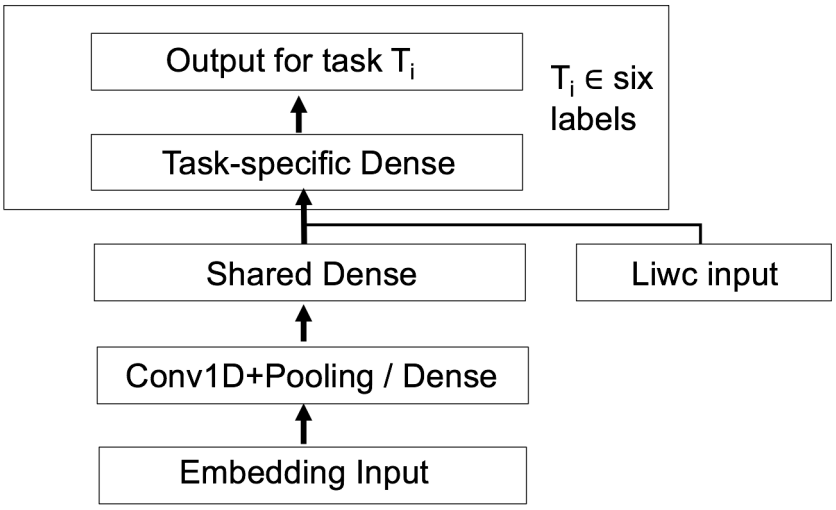


Figure 5.5: Using labels from LIWC 2015 as an additional input

5.5.5 MTL with LIWC as Additional Output

Then, in the next step, we want to use the generated labels from LIWC as additional outputs, i.e., auxiliary tasks, of the model. Multi-task learning is an algorithm that has dataset requirements – the data must have multiple annotated labels. Here, LIWC brings to us 92 new annotated labels. Regardless of the quality of the automatic machine annotation, this at least makes our dataset contains much more labels than the original one.

The structure of the model is shown in Fig. 5.6. We also want to see the difference in the results between using the LIWC labels as input or output. The result will be very interesting to us because they can show the strengths of the multi-task learning mechanism.

[2]https://liwc.wpengine.com/

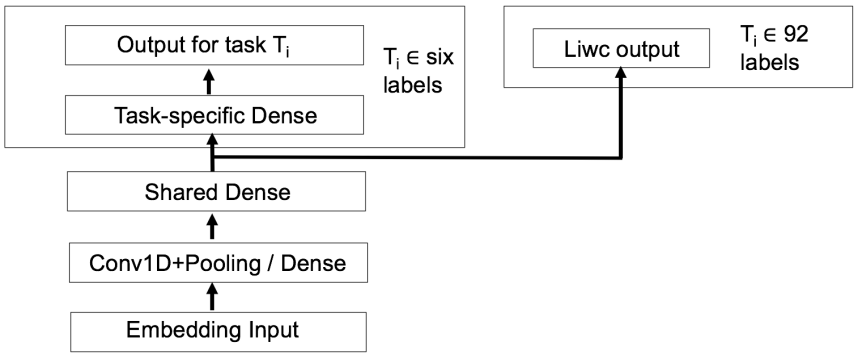


Figure 5.6: Using labels from LIWC 2015 as additional output (auxiliary tasks)

5.5.6 MTL with Basic Grouping

Let us look back at the definition of the six labels in this task. There are two labels, *Information_disclosure* and *Emotional_disclosure*, in category "Disclosure", for the disclosure of information and emotion separately; while in category "Supportiveness", *Information_support* and *Emotion_support* have a similar definition, but this time they are for supportiveness. Only *General_support* is a label which classifies quotes and catchphrases. Label *Support* generally covers all other three labels in category supportiveness.

In the task-specific parts of previous models, task-specific layers of all six labels are mutually non-overlapping. The bottom-most layers in task-specific parts of six tasks are connected to the same shared dense layer. However, if considering the definition and relationship among those labels, it is improper to align those task-specific layers horizontally in apple-pie order.

For example, *Information_disclosure* and *Emotional_disclosure* should be separated from other four labels because only these two refer to disclosure. Next, although the label *General_support* is classified into the category supportiveness, it pays more attention to classifying catchphrases instead of showing support. So *General_support* should also be separated from others. Finally, *Support*, *Information_support* and *Emotion_support* contain similar information, therefore we can group them together.

Based on those observations, an adjusted MTL structure is shown in Fig. 5.7.

Instead of all six task-specific parts connecting to the same shared layer, this time, there are three groups of labels which are connected to the same shared dense layer, whereas in each group, the labels are connected to group-specific shared layers: *Group-specific Dense 1,2,3*. In this way, the information flowing from the bottom to the top can be refined and specialized the second time. Group-specific layers share some functions and responsibilities of the lower shared layers. The topmost layers can be considered as "task-specific group-specific layers" now.

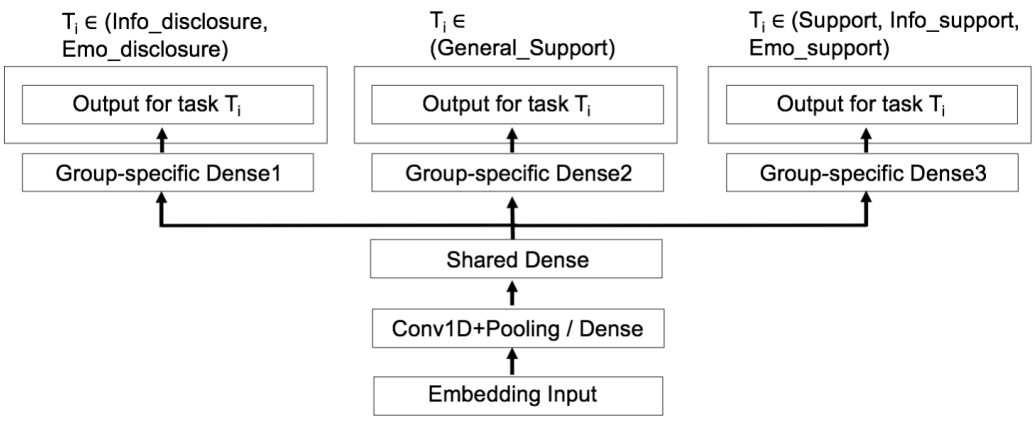


Figure 5.7: MTL structure with basic grouping

5.5.7 Venn-diagram-based Fragment MTL Model

The idea used in the previous section is simple and intuitive: the original MTL deep neural network can be seen as a multi-way tree, which contains one root and six leaves. Information flows from the bottom (input layer) to the top (output layer) and forks at the same level. In contrast, during training, information flows from the top to the bottom. By adjusting where the branches fork off and rectifying the information flows, the tree can represent the task more accurately.

Nevertheless, the grouping in 5.5.6 is based on the label description, which is arbitrary. We want to know the real relationship between six labels in the task. We are using Venn diagrams for this purpose.

A Venn diagram [doi, 1880] is a simple diagram used to represent unions and intersections of sets. However, based on the number of sets, it could be extremely complicated, which we will see below.

As similar tasks have similar patterns, we want their task-specific layers to be at a closer position compared with other tasks in the model. For example, among the six labels of our shared task, it is easy to image that the label *Support* has a strong relationship with the label *General_support*, *Info_support* and *Emo_support*, as they all refer to something about *Support*. Considering each label as a set, which contains entries in the training data where the corresponding label is *1*, the relationship among four labels can be described by the Venn diagram in Fig. 5.8. The numbers on the graph show the size of intersections among the sets.

From Fig. 5.8, we can see that the label *Support* covers almost all the cases in the other three sets, except for some trivial cases. Then how to reflect this relationship in the neural network? Because the label *Support* covers a more general concept, it should be treated as a lower layer. The other three labels refine information from the *Support* layer, using a part of its information, while sharing some neurons (i.e., a section of the layer) between themselves, as shown in Fig. 5.9. The bottom part of the Fig. 5.9 is a large dense layer,

which is split into several parts; we call it a fragment layer. Above the fragment layer in Fig. 5.9 are territories of four labels; this means that the label-corresponding task-specific layers will only connect to their specific territories (neurons).

Each label's territory has some overlap with other labels' territories, and the label *Support* occupies the whole layer, from the leftmost node to the rightmost node.

The example above is only for four labels. Nevertheless, *Support* can contain all the other three, which means that the intersections only appear among three layers. What about the six labels in our task? The Venn diagram is far less clear, as shown in Fig. 5.10. Discarding the pieces in the figure whose size is too small (intersections comprised of less than ten instances), there are in total 31 major intersections in the Venn diagram. For six-labels, each label is comprised of 15, 16, 28, 12, 12, and 15 intersections, respectively.

Roughly, from the bottom to the top, the network contains the input layer, shared hidden layers, task-specific layers and task-specific outputs. The connection between the shared hidden layers and task-specific layers is based on the fragments and the Venn diagrams, as shown above. As we have two types of embeddings, of shape (12860×1024) and shape $(12860 \times 34 \times 1024)$, we tried two types of models in the experiments.

For the data with shape (12860×1024), from the bottom to the top, we have an input layer, fragment dense layers, concatenate layers and output layers. As we mentioned above, the embedding layer is not included in the model because the word embeddings are already generated in pre-processing, using bert-as-service.

For the data with shape $(12860 \times 34 \times 1024)$, from the bottom to the top, the model is composed of an input layer, bidirectional LSTM layer, dense fragment layers, concatenate layer, attention layer [Vaswani et al., 2017], flatten layer and output layer.

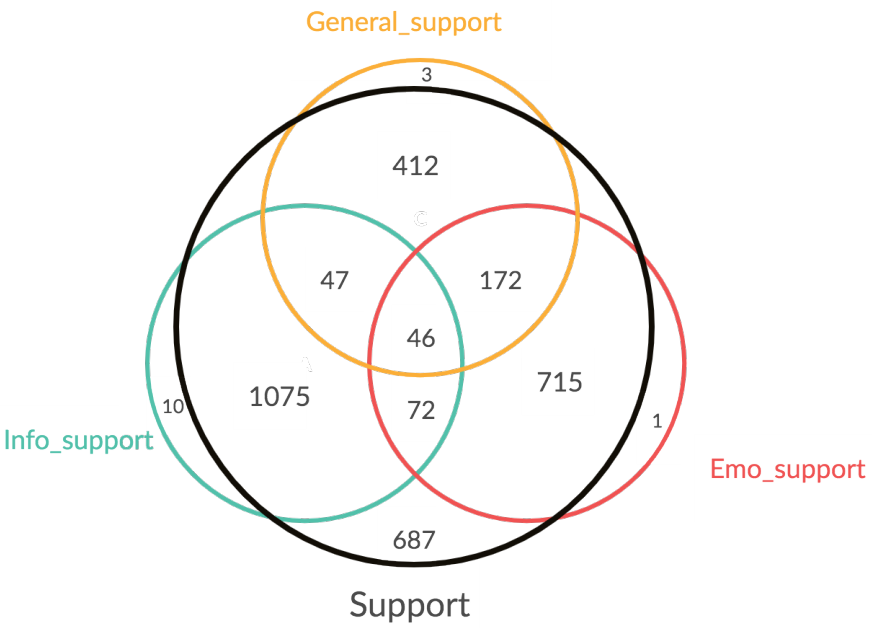


Figure 5.8: Venn diagram for label *Support*, *General_support*, *Info_support* and *Emo_support*

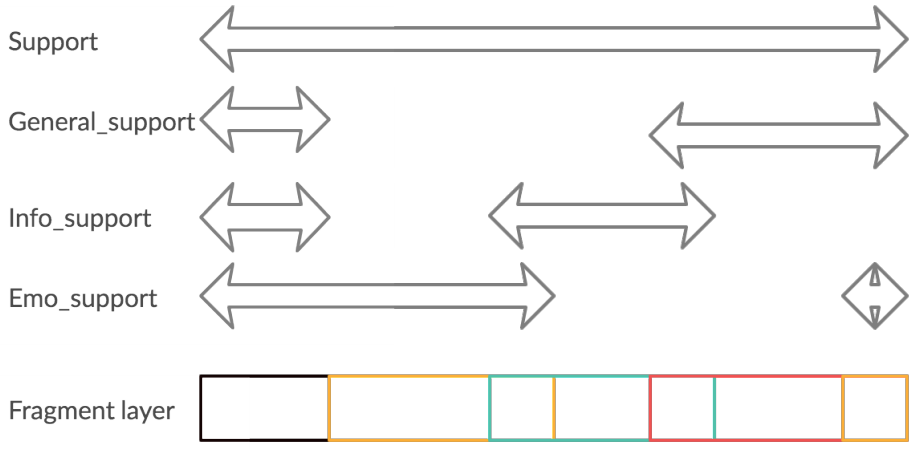


Figure 5.9: An example showing how fragment layer works for *Support*, *General_support*, *Info_support* and *Emo_support*

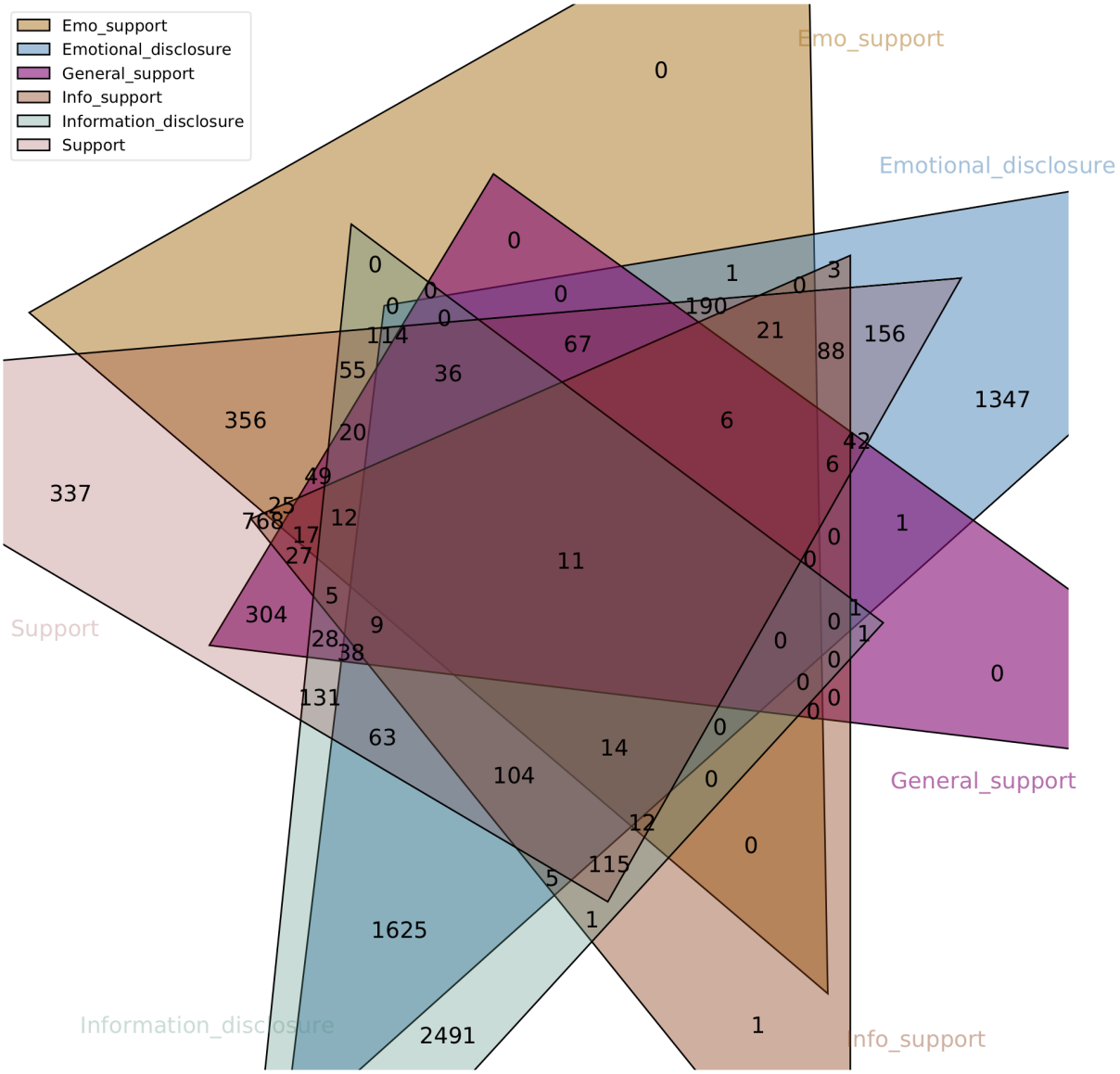


Figure 5.10: Venn diagram for all six labels

5.6 Experimental Results and Evaluation

During training, we use mini-batch gradient descent with batch size 256 and the Adam optimizer [Kingma and Ba, 2014] is used with a learning rate of 0.00002. The loss function is *binary cross-entropy* and activation function used in most of the models are *leaky Relu* [Xu et al., 2015], except the output layers, which use the *sigmoid* function.

All the results below are based on the K-fold cross-validation on the training data, with k equals 5[3], except for the results of the models submitted to the shared task for which we also report results on the test data (plus a few more results that we report of the test data that was made available after the shared task with revised labels).

There are two different embedding input files generated by bert-as-service, whose shapes are (12860×1024) and $(12860 \times 34 \times 1024)$. Most of the models have been tested on the former one with shape (12860×1024), while because of the time and memory requirements, only a few selected models are tested on latter one $(12860 \times 34 \times 1024)$.

Although we put the results on all metrics in the following tables, when we say a model is better than another, we mean that the F1-score of the model is better than that of the other model. We decided to focus of the F1-score for our model selection because the the F1-score combines precision and recall that are both important for our task.

5.6.1 Basic STL Model

Before looking at the results of multi-task learning models, let us look at the results of a single-task learning model. The STL model structure is very similar to the *Basic MTL Model* 5.5.1, except for the output layer, which only contains one node. This means that no multi-task learning mechanism is involved in the structure. We run this model on the 6 labels separately and obtained the results shown in Table 5.6.1.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6756	0.4791	0.6523	0.5522
Information_disclosure	0.6798	0.5746	0.6581	0.6130
Support	0.7906	0.5588	0.7599	0.6439
General_support	0.7467	0.1302	0.6687	0.2180
Info_support	0.7834	0.2857	0.7952	0.4204
Emo_support	0.8216	0.2405	0.7097	0.3592
Average	0.7496	0.3781	0.7073	0.4677

Table 5.3: Results of a basic STL model.

[3]We were not able to run with k=10 due to the time needed for training.

5.6.2 Basic MTL Model

Table 5.4 shows the results of the baseline MTL model on the input embedding with shape (12860 × 1024). Even though the model structure is simple, the results are reasonable.

We can see that among the six labels, the label *General_support* gets the worst result. This was expected considering the imbalance of the data for this label, where the ratio of positive cases to negative cases is 680 : 12,180. The *Support* label reaches 64% in terms of F1-score, but for the other supportiveness labels, *Info_support* and *Emo_support*, the results are not as good as for the generic label *Support*.

Compared with the results of STL model in 5.6.1, the average results on all metrics are improved with the help of multi-tasking. Interestingly, if we look deep into the F1-score of the six labels, we see that it improved for *Emotional_disclosure*, *Support*, *General_support* and *Emo_support*, and it stayed the same for *Information_disclosure*, and *Info_support*. This means that *Information_disclosure* and *Info_support* rarely get a benefit from MTL, whereas other labels do, possibly because the other labels contain no additional information useful for for *Information_disclosure* and *Info_support*, or because additional information cannot be passed through the model's structure. But from an overall perspective, the model performance on the whole task is improved, which is what we wanted to prove.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6798	0.4834	0.6470	0.5534
Information_disclosure	0.6810	0.5765	0.6543	0.6129
Support	0.7901	0.5573	0.7661	0.6452
General_support	0.7562	0.1358	0.6741	0.2260
Info_support	0.7833	0.2856	0.7957	0.4203
Emo_support	0.8230	0.2434	0.7157	0.3632
Average	0.7522	0.3803	0.7088	0.4701

Table 5.4: Result of Basic MTL Model.

5.6.3 Intermediate MTL Model with Conv1D and PRelu

In this section, we want to see the results from the intermediate MTL models and the performance difference between the two activation functions, leaky-Relu and PRelu.

The results of the model with leaky-Relu on the input embeddings with shape (12860 × 1024) are shown in the table 5.5, while for PRelu, the results are shown in table 5.6.

Both models have slightly better results than the baseline, even though the difference is small. Between leaky-Relu and PRelu, we can see that the model with PRelu achieves marginally better accuracy and F1-score. But, as the advantage of PRelu is too small to be considered useful, all the other models we built use only leaky-Relu and sigmoid as activation functions, which are far easier to implement compared with the PRelu activation function.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6823	0.4875	0.6669	0.5626
Information_disclosure	0.6829	0.5781	0.6624	0.6172
Support	0.7964	0.5667	0.7786	0.6558
General_support	0.7606	0.1406	0.6902	0.2335
Info_support	0.7879	0.2921	0.8053	0.4286
Emo_support	0.8264	0.2543	0.7478	0.3788
Average	0.7561	0.3866	0.7252	0.4794

Table 5.5: The result of intermediate MTL model with Conv1D and leaky-Relu.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6758	0.4811	0.6769	0.5611
Information_disclosure	0.6821	0.5770	0.6658	0.6177
Support	0.7975	0.5691	0.7765	0.6565
General_support	0.7781	0.1486	0.6759	0.2434
Info_support	0.7888	0.2939	0.8100	0.4312
Emo_support	0.8207	0.2478	0.7512	0.3722
Average	0.7572	0.3862	0.7261	0.4804

Table 5.6: The result of intermediate MTL model with Conv1D and PRelu.

5.6.4 MTL with a Combination of Three Embedding Methods

Before testing, we had high expectations for this model, because it contains three embedding methods and three information extraction layers. $(GloVe, ELMo, BERT) \times (Conv1D, BiLSTM, BiGRU)$. There are in total nine different combinations, plus the multi-task learning structure. It sould be a strong model, at least in theory.

As all three types of layers, Conv1D, BiLSTM and BiGRU, require three-dimensional inputs, this model is only tested on the input embedding with shape $(12860 \times 34 \times 1024)$.

After a few days for fine-tuning parameters to ensure that the model does not overfit, and a long time for training, we obtained the final results of this model, shown in table 5.7.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7022	0.5176	0.4380	0.4735
Information_disclosure	0.6655	0.5693	0.5716	0.5669
Support	0.8254	0.6709	0.5896	0.6267
General_support	0.9354	0.3110	0.1804	0.2261
Info_support	0.9007	0.4993	0.3890	0.4348
Emo_support	0.9306	0.5135	0.4375	0.4697
Average	0.8266	0.5136	0.4343	0.4663

Table 5.7: The results of model with a combination of three embedding methods.

Unfortunately, the results of the triple embedding model dropped steeply and were even

lower than the baseline model, which only uses the two-dimensional inputs. A reasonable explanation we can think of is that the performance of GloVe and ELMo is lagging behind that of the BERT embedding. The model does not only get the strong points from three embedding methods, but also their shortcomings. The results show that the strong points are overwhelmed by the shortcomings.

This taught us a useful lesson: to only focus on the best embedding method. Combining embedding methods did not prove to be useful. Therefore, all the following models we will present use only the BERT embeddings generated from bert-as-service.

5.6.5 MTL with LIWC as Additional Input

Next, we tried the commercial software LIWC 2015, which contains three versions of the LIWC dictionaries inside. It generates a corresponding analysis depending on the version of the dictionary chosen by the user. First, the result of the model with LIWC as additional input is shown in table 5.8. We can see from the table, that the results are slightly improved on all metrics, which means that the LIWC labels provide some help when used as extra input, but not as much as we expected.

A reason may be that BERT embedding already contains some of the features extracted by LIWC. Therefore only the new/distinct features help the model to classify better, while doubling the number of similar features cannot increase the model's performance.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6883	0.4941	0.6569	0.5637
Information_disclosure	0.6848	0.5807	0.6611	0.6181
Support	0.7942	0.5630	0.7794	0.6536
General_support	0.7566	0.1396	0.6973	0.2324
Info_support	0.7919	0.2959	0.8024	0.4323
Emo_support	0.8246	0.2504	0.7458	0.3748
Average	0.7567	0.3873	0.7238	0.4792

Table 5.8: The results of the model with LIWC as additional input.

5.6.6 MTL with LIWC as Additional Output

The previous section showed that using the LIWC annotations as additional input does not help much. So, what about treating them as additional outputs in the MTL model? Even if LIWC labels cannot provide extra information beyond the BERT embeddings, auxiliary labels can still help the multi-task learning process by enhancing the information flow and preventing the model from overfitting.

The results are shown in table 5.9. We can see that the improvement is marginal.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6824	0.4873	0.6600	0.5601
Information_disclosure	0.6854	0.5825	0.6611	0.6185
Support	0.7938	0.5647	0.7775	0.6529
General_support	0.7626	0.1415	0.6875	0.2345
Info_support	0.7997	0.3061	0.7995	0.4418
Emo_support	0.8224	0.2480	0.7411	0.3711
Average	0.7577	0.3883	0.7211	0.4798

Table 5.9: The result of model with LIWC as additional output.

5.6.7 MTL with Basic Grouping

The results of the MTL model with basic grouping is shown in table 5.10. Even though the advantage is small, the results are better than for all the models above, which confirms our assumption that particularizing the branches in the network can help the MTL training.

We only show the results of the model when the labels are split into three groups, based on the disclosure and supportiveness:

- Group1: Information_disclosure, Emotional_disclosure
- Group2: General_support
- Group3: Support, Info_support, Emo_support

We tried several different ways of grouping, for example:

Grouping based on whether the label is informational or emotional, or neither:

- Group1: Information_disclosure, Info_support
- Group2: Emotional_disclosure, Emo_support
- Group3: Support, General_support

Grouping based on all possible relationship, and concatenating them together later:

- Group1: Information_disclosure, Emotional_disclosure
- Group2: Information_disclosure, Support, Info_support
- Group3: Emotional_disclosure, Emo_support
- Group4: Support, General_support, Info_support, Emo_support
- Group5: General_support, Emo_support

All these grouping approaches lead to similar results with the first one, so they are omitted here.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6841	0.4904	0.6664	0.5636
Information_disclosure	0.6836	0.5783	0.6680	0.6196
Support	0.7991	0.5718	0.7765	0.6583
General_support	0.7620	0.1432	0.7009	0.2376
Info_support	0.8000	0.3051	0.7986	0.4412
Emo_support	0.8259	0.2509	0.7371	0.3741
Average	0.7591	0.3899	0.7246	0.4824

Table 5.10: The results of the MTL model with basic grouping.

5.6.8 Venn-diagram-based Fragment MTL Model

When looking again at the grouping methods used in the previous section 5.6.7, we think that all the groupings make sense, but none of them can cover all the relationships between the six labels completely, not even the last one that has five groups.

Using *Information_disclosure* and *Emo_support* as an example, apparently there is no connection between the two labels according to the label descriptions. But if we look at the Venn diagram in Fig. 5.11, we can find that over one third of the entries in *Emo_support* are also included in *Information_disclosure*. So there is a degree of relationships between these two labels. Then, we need to extend the relation extraction method using the Venn diagram, from two labels to all the six labels (pairwise), as shown in Fig. 5.10. We implement a new model based on it.

In this model, we tested both embedding inputs, with shape (12860×1024) and $(12860 \times 34 \times 1024)$.

For the embedding file with shape (12860×1024), the results are shown in table 5.11. We can see that, given the same embedding input, the model based on the Venn diagram has the best F1-score among all the models mentioned before.

Furthermore, compared with the input (12860×1024), the three-dimensional input $(12860 \times 34 \times 1024)$ contains more detailed information in embeddings and should have better results than the two-dimensional input. This result is shown in the table 5.12. Though from the table, we can see that the Venn based model with three-dimensional input has a lower F1-score for *Supportiveness* labels, but much higher F1-score for *Disclosure* labels. Finally, we conclude that the average F1-score is with 1% higher for the three-dimensional input than for the two-dimensional input.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6941	0.5043	0.5791	0.5360
Information_disclosure	0.6934	0.6203	0.5568	0.5819
Support	0.8171	0.6197	0.7181	0.6619
General_support	0.8003	0.1540	0.6116	0.2453
Info_support	0.8449	0.3565	0.6921	0.4691
Emo_support	0.8676	0.3088	0.6916	0.4256
Average	0.7862	0.4273	0.6416	0.4866

Table 5.11: The result of Venn diagram based fragment MTL model on embeddings (12860×1024).

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.6314	0.4405	0.7803	0.5631
Information_disclosure	0.6645	0.5457	0.7497	0.6317
Support	0.7597	0.5138	0.8611	0.6436
General_support	0.7547	0.1573	0.7832	0.2620
Info_support	0.8453	0.3492	0.7154	0.4693
Emo_support	0.8348	0.2668	0.7637	0.3954
Average	0.7484	0.3789	0.7756	0.4942

Table 5.12: The result of Venn-diagram-based fragment MTL model on embeddings ($12860 \times 34 \times 1024$).

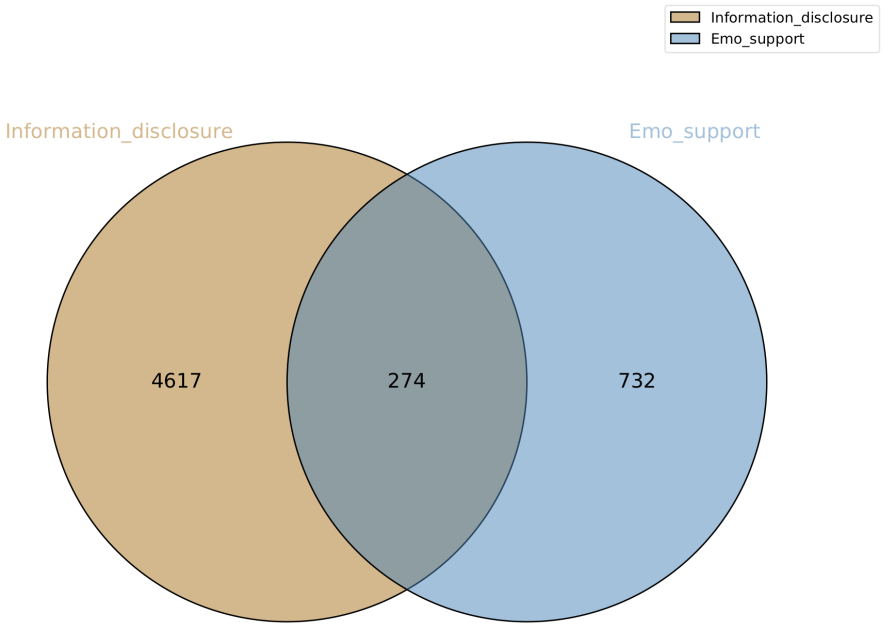


Figure 5.11: Venn diagram of *Emo_support* and *Information_disclosure*.

5.6.9 Results on the Six Labels across Models

To compare the results on the six labels across the models more clearly, we rearranged the tables above and put the results of each label into individual tables.

Emotional_disclosure

Table 5.13 shows the results of all the models on the label *Emotional_disclosure*.

	Accuracy	Precision	Recall	F1-score
Basic STL	0.6756	0.4791	0.6523	0.5522
Basic MTL	0.6798	0.4834	0.6470	0.5534
Intermediate MTL with Conv1D(leaky_relu)	0.6823	0.4875	0.6669	0.5626
Intermediate MTL with Conv1D(PRelu)	0.6758	0.4811	0.6769	0.5611
MTL with 3 embedding	0.7022	0.5176	0.4380	0.4735
MTL with LIWC input	0.6883	0.4941	0.6569	0.5637
MTL with LIWC output	0.6824	0.4873	0.6600	0.5601
MTL with basic grouping	0.6841	0.4904	0.6664	0.5636
Venn-based MTL	0.6941	0.5043	0.5791	0.5360

Table 5.13: Models results on the label *Emotional_disclosure*.

Information_disclosure

Table 5.14 shows the results of all the models on the label *Information_disclosure*.

	Accuracy	Precision	Recall	F1-score
Basic STL	0.6798	0.5746	0.6581	0.6130
Basic MTL	0.6810	0.5765	0.6543	0.6129
Intermediate MTL with Conv1D(leaky_relu)	0.6829	0.5781	0.6624	0.6172
Intermediate MTL with Conv1D(PRelu)	0.6821	0.5770	0.6658	0.6177
MTL with 3 embedding	0.6655	0.5693	0.5716	0.5669
MTL with LIWC input	0.6848	0.5807	0.6611	0.6181
MTL with LIWC output	0.6854	0.5825	0.6611	0.6185
MTL with basic grouping	0.6836	0.5783	0.6680	0.6196
Venn-based MTL	0.6934	0.6203	0.5568	0.5819

Table 5.14: Models results on the label *Information_disclosure*.

Support

Table 5.15 shows the results of all the models on the label *Support*.

	Accuracy	Precision	Recall	F1-score
Basic STL	0.7906	0.5588	0.7599	0.6439
Basic MTL	0.7901	0.5573	0.7661	0.6452
Intermediate MTL with Conv1D(leaky_relu)	0.7964	0.5667	0.7786	0.6558
Intermediate MTL with Conv1D(PRelu)	0.7975	0.5691	0.7765	0.6565
MTL with 3 embedding	0.8254	0.6709	0.5896	0.6267
MTL with LIWC input	0.7942	0.5630	0.7794	0.6536
MTL with LIWC output	0.7938	0.5647	0.7775	0.6529
MTL with basic grouping	0.7991	0.5718	0.7765	0.6583
Venn-based MTL	0.8171	0.6197	0.7181	0.6619

Table 5.15: Models results on the label *Support*.

General_support

Table 5.16 shows the results of all the models the on label *General_support*.

	Accuracy	Precision	Recall	F1-score
Basic STL	0.7467	0.1302	0.6687	0.2180
Basic MTL	0.7562	0.1358	0.6741	0.2260
Intermediate MTL with Conv1D(leaky_relu)	0.7606	0.1406	0.6902	0.2335
Intermediate MTL with Conv1D(PRelu)	0.7781	0.1486	0.6759	0.2434
MTL with 3 embedding	0.9354	0.3110	0.1804	0.2261
MTL with LIWC input	0.7566	0.1396	0.6973	0.2324
MTL with LIWC output	0.7626	0.1415	0.6875	0.2345
MTL with basic grouping	0.7620	0.1432	0.7009	0.2376
Venn-based MTL	0.8003	0.1540	0.6116	0.2453

Table 5.16: Models results on the label *General_support*.

Info_support

Table 5.17 shows the results of all the models on the label *Info_support*.

	Accuracy	Precision	Recall	F1-score
Basic STL	0.7834	0.2857	0.7952	0.4204
Basic MTL	0.7833	0.2856	0.7957	0.4203
Intermediate MTL with Conv1D(leaky_relu)	0.7879	0.2921	0.8053	0.4286
Intermediate MTL with Conv1D(PRelu)	0.7888	0.2939	0.8100	0.4312
MTL with 3 embedding	0.9007	0.4993	0.3890	0.4348
MTL with LIWC input	0.7919	0.2959	0.8024	0.4323
MTL with LIWC output	0.7997	0.3061	0.7995	0.4418
MTL with basic grouping	0.8000	0.3051	0.7986	0.4412
Venn-based MTL	0.8449	0.3565	0.6921	0.4691

Table 5.17: Models results on the label *Info_support*.

Emo_support

Table 5.18 shows the results of all the models on the label *Emo_support*.

	Accuracy	Precision	Recall	F1-score
Basic STL	0.8216	0.2405	0.7097	0.3592
Basic MTL	0.8230	0.2434	0.7157	0.3632
Intermediate MTL with Conv1D(leaky_relu)	0.8264	0.2543	0.7478	0.3788
Intermediate MTL with Conv1D(PRelu)	0.8207	0.2478	0.7512	0.3722
MTL with 3 embedding	0.9306	0.5135	0.4375	0.4697
MTL with LIWC input	0.8246	0.2504	0.7458	0.3748
MTL with LIWC output	0.8224	0.2480	0.7411	0.3711
MTL with basic grouping	0.8259	0.2509	0.7371	0.3741
Venn-based MTL	0.8676	0.3088	0.6916	0.4256

Table 5.18: Models results on the label *Emo_support*.

Average

Table 5.19 shows the average results of all the models on all labels.

	Accuracy	Precision	Recall	F1-score
Basic STL	0.7496	0.3781	0.7073	0.4677
Basic MTL	0.7522	0.3803	0.7088	0.4701
Intermediate MTL with Conv1D(leaky_relu)	0.7561	0.3866	0.7252	0.4794
Intermediate MTL with Conv1D(PRelu)	0.7572	0.3862	0.7261	0.4804
MTL with 3 embedding	0.8266	0.5136	0.4343	0.4663
MTL with LIWC input	0.7567	0.3873	0.7238	0.4792
MTL with LIWC output	0.7577	0.3883	0.7211	0.4798
MTL with basic grouping	0.7591	0.3899	0.7246	0.4824
Venn-based MTL	0.7862	0.4273	0.6416	0.4866

Table 5.19: Average models results on all labels.

5.6.10 Model Results on the Test Data

Luckily, just several days before the submission of this thesis, the labels for the shared task's test data were released by the organising committee. Several updated labels were introduced into the test set, so we decided to train our models again and test it on the latest version of the labelled test data to see the changes. The labelled test set released contains 8 labels, which can be separated into 2 groups. The first one contains: *SharedTask_Emotional_disclosure*, *SharedTask_Information_disclosure*, *SharedTask_Emo_support*, *SharedTask_Info_support*; while the second group contains: *Emotional_disclosure*, *Information_disclosure*, *Emo_support*, *Info_support*. According to the explanation from the committee, the first group, whose labels start with *SharedTask*, are the labels used in the CL-Aff Shared Task 2020 competition; after the competition, the committee found some errors in the test set so they renewed some labels, which are then saved as new labels in the second

group. We can see that only 4 out of 6 labels are released, which means these 4 labels are really valued by the committee. We will compare the results from these 4 labels in this section.

Therefore, after all the experiments above, we reran our code to evaluate the model performance on the test sets. The models are trained on 80% of the training data, with 20% of the training data used for validation. The results generated by our models on the test data are then compared with the test data labels.

We tested the model performance on both the *Shared Task* labels and the latest labels. Surprisingly, the results are significantly improved on the latest labels, which is even higher than the training results. It is confusing to us because generally the testing score should be lower than the training score. We do not know how exactly the labels are adjusted by the committee, so the results below of the latest labels are only for your information, but will not be discussed deeply.

Basic STL Model Results on the Test Set

The basic STL model results on the shared task's test set are shown in table 5.20 and the results on the latest labels are shown in table 5.21.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5346	0.7008	0.5972	0.6449
Information_disclosure	0.5012	0.6353	0.4919	0.5545
Emo_support	0.547	0.2886	0.348	0.3155
Info_support	0.6088	0.2377	0.3043	0.2669
Average	0.5479	0.4656	0.4353	0.4454

Table 5.20: Basic STL model results on the shared task's test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7726	0.7479	0.8568	0.7987
Information_disclosure	0.7636	0.8809	0.7072	0.7845
Emo_support	0.794	0.4771	0.9113	0.6263
Info_support	0.8016	0.5007	0.7545	0.6019
Average	0.7829	0.6516	0.8074	0.7028

Table 5.21: Basic STL model results on the latest labelled test set.

These results can be treated as baselines for the following model's results.

Basic MTL model results on the test set

The basic MTL model results on the shared task's test set are shown in table 5.22 and the results on the latest labels are shown in table 5.23.

For table 5.22, we can see that because of the dataset changes, the results are a little lower than those from table 5.6.2. For the latest labelled test set results in table 5.23, the results are improved a lot, which makes us think that the maybe committee found some labelling errors in the original test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5316	0.6970	0.5981	0.6437
Information_disclosure	0.4968	0.6310	0.4878	0.5502
Emo_support	0.5546	0.2968	0.3540	0.3229
Info_support	0.6028	0.2358	0.3111	0.2682
Average	0.5464	0.4651	0.4377	0.4463

Table 5.22: Basic MTL model results on the shared task's test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7692	0.7434	0.8575	0.7964
Information_disclosure	0.7588	0.8766	0.7026	0.7800
Emo_support	0.8080	0.4964	0.9377	0.6491
Info_support	0.7948	0.4896	0.7606	0.5957
Average	0.7827	0.6515	0.8146	0.7053

Table 5.23: Basic MTL model results on the latest labelled test set.

Intermediate MTL with leaky-Relu model results on the test set

The intermediate MTL with leaky-Relu model results on the shared task's test set are shown in table 5.24 and the results on the latest labels are shown in table 5.25.

Among the four labels, only the resulst for *Emotional_disclosure* is increased compared with the training results in Table 5.6.3, while for the other three labels, the results are decreased in table 5.24. In table 5.25, all the results are greatly increased, and they are higher than those of the basic MTL model.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5388	0.7007	0.6080	0.6510
Information_disclosure	0.5052	0.6323	0.5157	0.5681
Emo_support	0.5526	0.2942	0.3513	0.3203
Info_support	0.6148	0.2319	0.2795	0.2535
Average	0.5529	0.4648	0.4386	0.4482

Table 5.24: Intermediate MTL with leaky-Relu model results on the shared task's test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7696	0.7410	0.8644	0.7980
Information_disclosure	0.7676	0.8655	0.7318	0.7931
Emo_support	0.8084	0.4969	0.9398	0.6501
Info_support	0.8136	0.5220	0.7404	0.6123
Average	0.7898	0.6564	0.8191	0.7134

Table 5.25: Intermediate MTL with leaky-Relu model results on the latest labelled test set.

Intermediate MTL with PRelu model results on test set

The intermediate MTL with PRelu model result on the shared task's test set is shown in table 5.26 and the result on the latest labels is shown in table 5.27. We can see the results are very similar with the results in the intermediate MTL with leaky-Relu model in 5.6.10.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5362	0.6998	0.6034	0.6480
Information_disclosure	0.4930	0.6303	0.4754	0.5420
Emo_support	0.5486	0.2891	0.3460	0.3150
Info_support	0.6052	0.2362	0.3077	0.2673
Average	0.5457	0.4638	0.4331	0.4431

Table 5.26: Intermediate MTL with PRelu model results on the SharedTask labelled test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7822	0.7529	0.8727	0.8084
Information_disclosure	0.7602	0.8874	0.6941	0.7789
Emo_support	0.8092	0.4981	0.9440	0.6521
Info_support	0.8008	0.4993	0.7656	0.6044
Average	0.7881	0.6594	0.8191	0.7110

Table 5.27: Intermediate MTL with PRelu model results on the latest labelled test set.

MTL with a combination of three embedding methods model results on test set

The MTL with a combination of three embedding methods model result on the shared task's test set is shown in table 5.28 and the result on the latest labels is shown in table 5.29. Similar with the decline in the training data, the results on the test data here are lower than other model results, too. And the results on shared task labels are even lower than the basic MTL model.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5124	0.6996	0.5449	0.6126
Information_disclosure	0.4564	0.6332	0.3293	0.4333
Emo_support	0.6106	0.2836	0.1953	0.2313
Info_support	0.6378	0.2322	0.2376	0.2349
Average	0.5543	0.4622	0.3268	0.3780

Table 5.28: MTL with a combination of three embedding methods model results on the SharedTask labelled test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7668	0.7660	0.8021	0.7836
Information_disclosure	0.6928	0.9592	0.5173	0.6721
Emo_support	0.8996	0.7154	0.7804	0.7465
Info_support	0.8318	0.5639	0.6791	0.6162
Average	0.7977	0.7511	0.6947	0.7046

Table 5.29: MTL with a combination of three embedding methods model results on the latest labelled test set.

MTL with LIWC as additional input model results on test set

The MTL with LIWC as additional input model result on the shared task's test set is shown in table 5.30 and the result on the latest labels is shown in table 5.31.

Unfortunately, even with the help of additional input from LIWC, we cannot get the improvement we want. The results shown in both tables are very close to the intermediate model in 5.6.10.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5260	0.7043	0.5690	0.6295
Information_disclosure	0.4988	0.6353	0.4830	0.5488
Emo_support	0.5528	0.2937	0.3493	0.3191
Info_support	0.5956	0.2327	0.3171	0.2685
Average	0.5433	0.4665	0.4296	0.4415

Table 5.30: MTL with LIWC as additional input model results on the SharedTask labelled test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7692	0.7586	0.8237	0.7898
Information_disclosure	0.7592	0.8833	0.6964	0.7788
Emo_support	0.8110	0.5006	0.9430	0.6540
Info_support	0.7888	0.4806	0.7706	0.5920
Average	0.7820	0.6557	0.8084	0.7036

Table 5.31: MTL with LIWC as additional input model results on the latest labelled test set.

MTL with LIWC as additional output model results on test set

The MTL with LIWC as additional output model result on the shared task's test set is shown in table 5.32 and the result on the latest labels is shown in table 5.33.

Compared with the MTL with LIWC as additional input model in 5.6.10, LIWC as output at least increased the results on shared task labels. We can see that the average F1-score is higher than all other previous models, but still cannot beat the Venn-diagram-based model below.

	Accuracy	Precision	Pecall	F1-score
Emotional_disclosure	0.5686	0.7002	0.6826	0.6913
Information_disclosure	0.5026	0.6354	0.4967	0.5576
Emo_support	0.5356	0.2939	0.3907	0.3354
Info_support	0.6118	0.2397	0.3034	0.2678
Average	0.5547	0.4673	0.4683	0.4630

Table 5.32: MTL with LIWC as additional output model results on the shared task test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7582	0.7063	0.9255	0.8012
Information_disclosure	0.7586	0.8723	0.7069	0.7809
Emo_support	0.7730	0.4529	0.9535	0.6141
Info_support	0.8054	0.5071	0.7555	0.6069
Average	0.7738	0.6346	0.8354	0.7008

Table 5.33: MTL with LIWC as additional output model results on the latest labelled test set.

MTL with basic grouping model results on test set

The MTL with basic grouping model result on the shared task's test set is shown in table 5.34 and the result on the latest labels is shown in table 5.35.

It shows that the basic grouping cannot lead to an improvement in the test set. The reason may be the data shift in the test set, or the insufficiency of basic grouping.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5448	0.6978	0.6292	0.6617
Information_disclosure	0.4932	0.6364	0.4593	0.5335
Emo_support	0.5440	0.2903	0.3600	0.3214
Info_support	0.5986	0.2339	0.3145	0.2683
Average	0.5451	0.4646	0.4407	0.4462

Table 5.34: MTL with basic grouping model results on the shared task test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7680	0.7307	0.8856	0.8008
Information_disclosure	0.7576	0.9021	0.6750	0.7722
Emo_support	0.7998	0.4855	0.9535	0.6434
Info_support	0.7978	0.4946	0.7827	0.6062
Average	0.7808	0.6532	0.8242	0.7056

Table 5.35: MTL with basic grouping model results on the latest labelled test set.

Venn-diagram-based fragment MTL model results on test set

The Venn-diagram-based fragment MTL model result on the shared task's test set is shown in table 5.36 and the result on the latest labels is shown in table 5.37.

Venn-diagram-based model is the only one whose average F1-score has exceeded 0.48 on shared task labels and 0.72 on latest labels. We are glad to see that it can obtain the best results on the test set among all models, and the improvement is much more obvious than which in the training data.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.5788	0.7010	0.7058	0.7034
Information_disclosure	0.5456	0.6314	0.6726	0.6513
Emo_support	0.5612	0.2910	0.3220	0.3057
Info_support	0.6216	0.2403	0.2855	0.2609
Average	0.5768	0.4659	0.4965	0.4803

Table 5.36: Venn-diagram-based fragment MTL model results on the shared task test set.

	Accuracy	Precision	Recall	F1-score
Emotional_disclosure	0.7568	0.6988	0.9457	0.8037
Information_disclosure	0.7452	0.7632	0.8429	0.8011
Emo_support	0.8338	0.5349	0.9377	0.6812
Info_support	0.8184	0.5309	0.7425	0.6191
Average	0.7885	0.6320	0.8672	0.7263

Table 5.37: Venn-diagram-based fragment MTL model results on the latest labelled test set.

Average results across all models on the four labels

Again, to compare the results across the models, we align the average results of each model and put them into the same tables. The following two tables show the average results on the four labels of all models, on the shared task's test set and the latest labelled test set.

Table 5.38 shows the average results on the shared task's test set. Table 5.39 shows the average results on the latest labelled test set.

From these two tables, we can see that, except for the model *MTL with a combination of three embedding methods*, all the other models have a F1-score higher than 0.44 on the shared task's test set and higher than 0.70 on the latest labelled test set. Our best model, the *Venn-diagram-based fragment MTL model*, always has the best F1-score, on both the shared task's test set and the latest labelled test set.

	Accuracy	Precision	Recall	F1-score
Basic STL model	0.5479	0.4656	0.4353	0.4454
Basic MTL model	0.5464	0.4651	0.4377	0.4463
Intermediate MTL with leaky-Relu	0.5529	0.4648	0.4386	0.4482
Intermediate MTL with PRelu	0.5457	0.4638	0.4331	0.4431
MTL with a combination of 3 embeddings	0.5543	0.4622	0.3268	0.3780
MTL with LIWC input	0.5433	0.4665	0.4296	0.4415
MTL with LIWC output	0.5547	0.4673	0.4683	0.4630
MTL with basic grouping	0.5451	0.4646	0.4407	0.4462
Venn-diagram-based MTL	0.5768	0.4659	0.4965	0.4803

Table 5.38: Average results across models on the shared task's test set.

	Accuracy	Precision	Recall	F1-score
Basic STL model	0.7829	0.6516	0.8074	0.7028
Basic MTL model	0.7827	0.6515	0.8146	0.7053
Intermediate MTL with leaky-Relu	0.7898	0.6564	0.8191	0.7134
Intermediate MTL with PRelu	0.7881	0.6594	0.8191	0.7110
MTL with a combination of 3 embeddings	0.7977	0.7511	0.6947	0.7046
MTL with LIWC input	0.7820	0.6557	0.8084	0.7036
MTL with LIWC output	0.7738	0.6346	0.8354	0.7008
MTL with basic grouping	0.7808	0.6532	0.8242	0.7056
Venn-diagram-based MTL	0.7885	0.6320	0.8672	0.7263

Table 5.39: Average results across models on the latest labelled test set.

5.7 Comparison to Related Work

The overview paper from the shared task 2020 was released by [Jaidka et al., 2020], so we can compare our results with those of the other teams who participated.

The shared task organizers list two pairs of labels for comparison: *Emo_support* and *Info_support*; *Emotional_disclosure* and *Information_disclosure*. The other two labels, *Support* and *General_support* are not included in the overview paper.

Fig. 5.12 shows the accuracy score of *Emo_support* and *Info_support*. Our team, **UOTTAWA CANADA** ranked third among the six teams, but the scores of top three are very close to each other.

Fig. 5.13 shows the prediction accuracy for *Emotional_disclosure* and *Information_disclosure*. This time, our team's models performed the best among the six teams, and the gap between our score and the second best is bigger than the gap for the support labels.

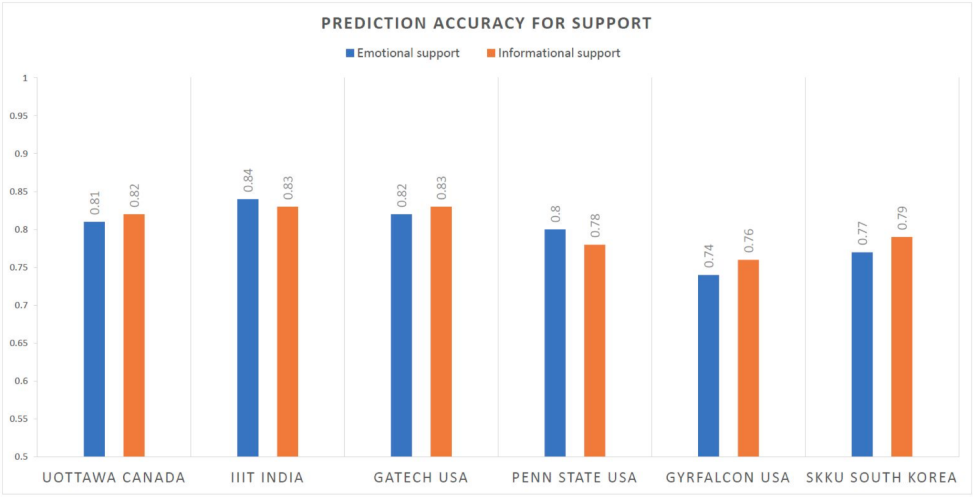


Figure 5.12: Prediction accuracy of the six teams on label *Emo_support* and *Info_support*.

The following are the models of the other five teams, from the left to the right in the tables 5.12 and 5.13.

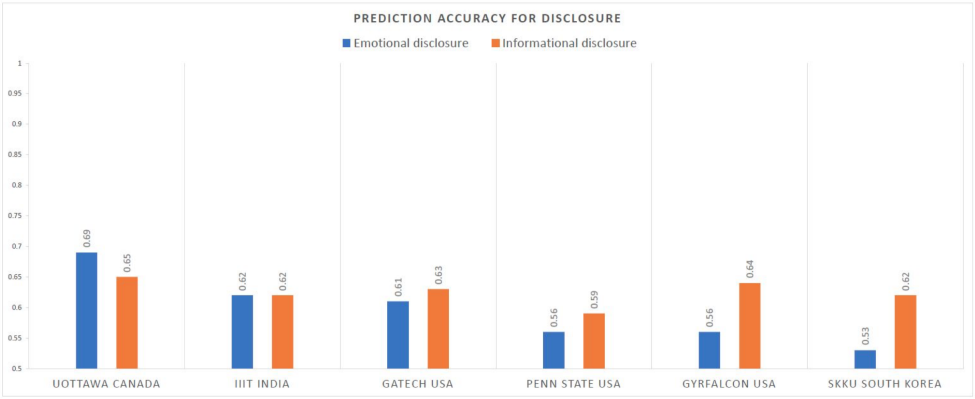


Figure 5.13: Prediction accuracy of the six teams on label *Emotional_disclosure* and *Information_disclosure*.

BERT-based Ensembles for Modeling Disclosure and Support in Conversational Social Media Text

The team **IIIT INDIA** introduced a predictive model that uses transfer learning in the form of pre-trained BERT-based models. They proposed an ensemble of two pre-trained models: $RoBERTa$ and $ALBERT$.

Fig. 5.14 depicts the proposed ensemble model. In this model, a sentence is processed in parallel by the $RoBERTa$ and $ALBERT$ models fine-tuned for predicting each label. The results from these base models are then combined using a weighted average ensemble technique to predict the final labels, which includes predictions for the six labels.

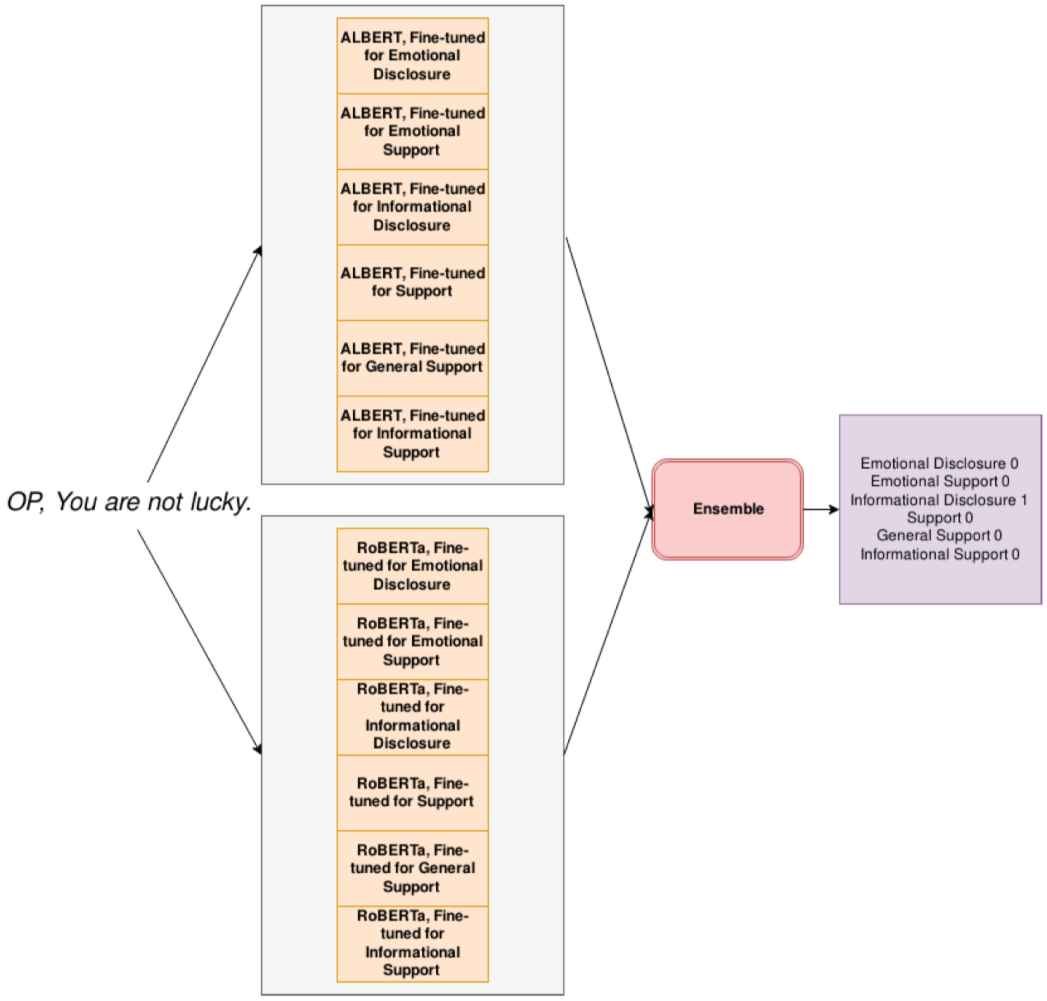


Figure 5.14: Architecture of the ensemble models which predict the label set denoting support and disclosure from the comment text, from the team **IIIT INDIA**.

Semi-supervised Models via Data Augmentation for Classifying Interactive Affective Responses

The team **GATECH USA** proposed semi-supervised models with data augmentation (SMDA) to classify the interactive affective responses. SMDA utilizes recent transformer-based models to encode each sentence and employs back translation techniques to paraphrase given sentences as augmented data. For the labeled sentences, data augmentations were performed to make the label distributions uniform and the models computed the supervised loss during the training process. For the unlabeled sentences, the authors explored self-training by regarding low-entropy predictions over unlabeled sentences as pseudo-labels, assuming high-confidence predictions as labeled data for training. This team further introduced consistency regularization as unsupervised loss after data augmentations on the unlabeled data, based on the assumption that the model should predict similar class distributions with the original unlabeled sentences as input and augmented

sentences as input.

In the supervised learning for the labeled sentences, for each labeled input sentence, this team used XLNet [Yang et al., 2019] to encode it into a hidden representation, and then passed it though a two-layer neural network to predict the class distributions.

Contextual Representation of Self-Disclosure and Supportiveness in Short Text

The team **PENN STATE USA** developed a multi-modal approach for the joint classification of self-disclosure and supportiveness in short texts. They took an ensemble approach for representation learning, leveraging BERT, LSTM, and CNN neural networks.

This team proposed two models. The first one is called *BERT model with fine-tuning*. They fine-tuned the BERT model using a Masked Language Model (LM) and the post and comment data provided in the unlabeled dataset. Their model structure is shown in Fig. 5.15.

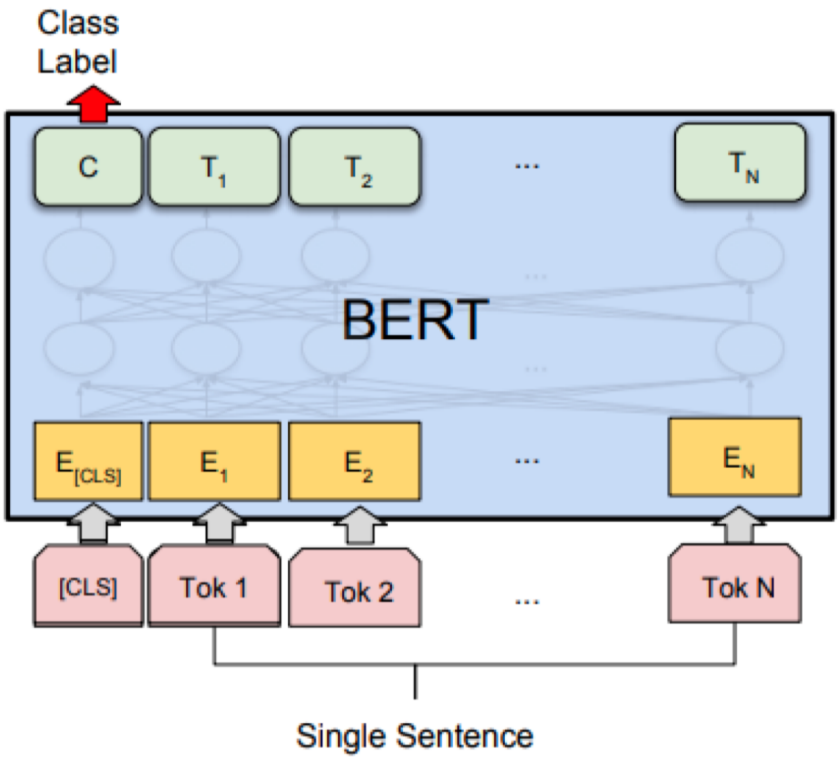


Figure 5.15: Sentence classification using Bidirectional Transformers, from the team **PENN STATE USA**.

The second model is called *CNN with BERT embedding*. The sentences were tokenized and word representations were generated from a BERT-based pre-trained model. The results were obtained from the output of the CNN. Fig. 5.16 shows the structure of this model.

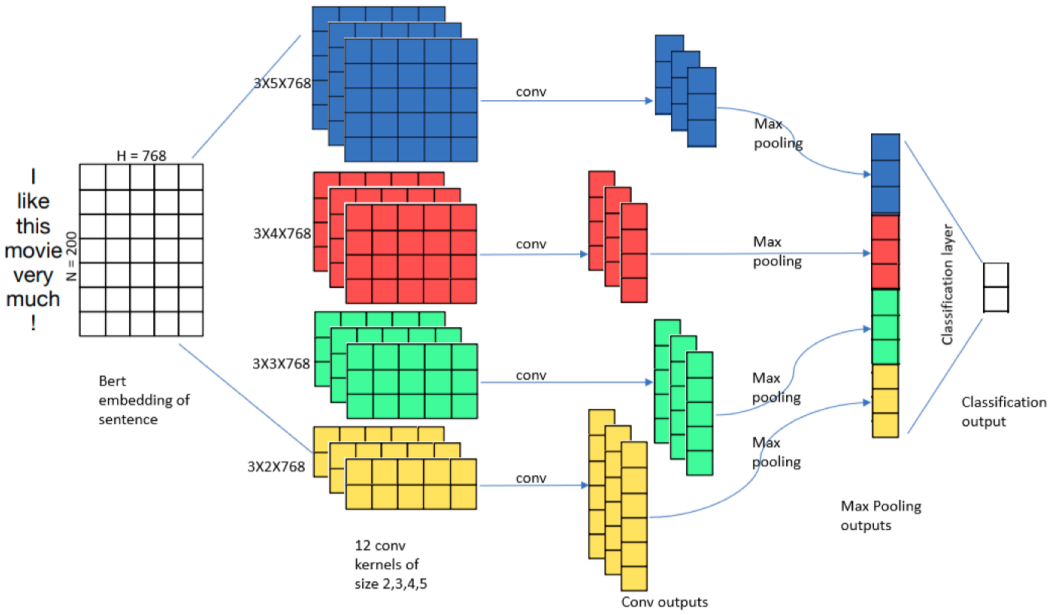


Figure 5.16: CNN model with BERT embedding, from the team **PENN STATE USA**.

Multi-modal Sentiment Analysis using Super Characters Method on Low-power CNN Accelerator Device

The team **GYRFALCON USA** proposed an interesting power-saving method. The authors combined the super-characters method with the convolutional neural network and deployed the model on CNN accelerating chips. Fig. 5.17 shows an overview of the super-characters method. From the graph, we can see that the sentence is converted into a picture. In this way, the authors bypass the need for word embeddings and use CNN, which is powerful in image processing, in order to deal with the texts.

The result is reasonable, considering that no word embedding method was used. One question here is the authors do not compare the results of the model on the chips with the model in the a regular environment, so we do not know how much the power-saving method affects the results. As shown in Fig. 5.17, long words and short words use the same size in the input. However, generally, long words tend to be more meaningful than short words. In CNN, the convolutional layer will blur the importance of the long words.

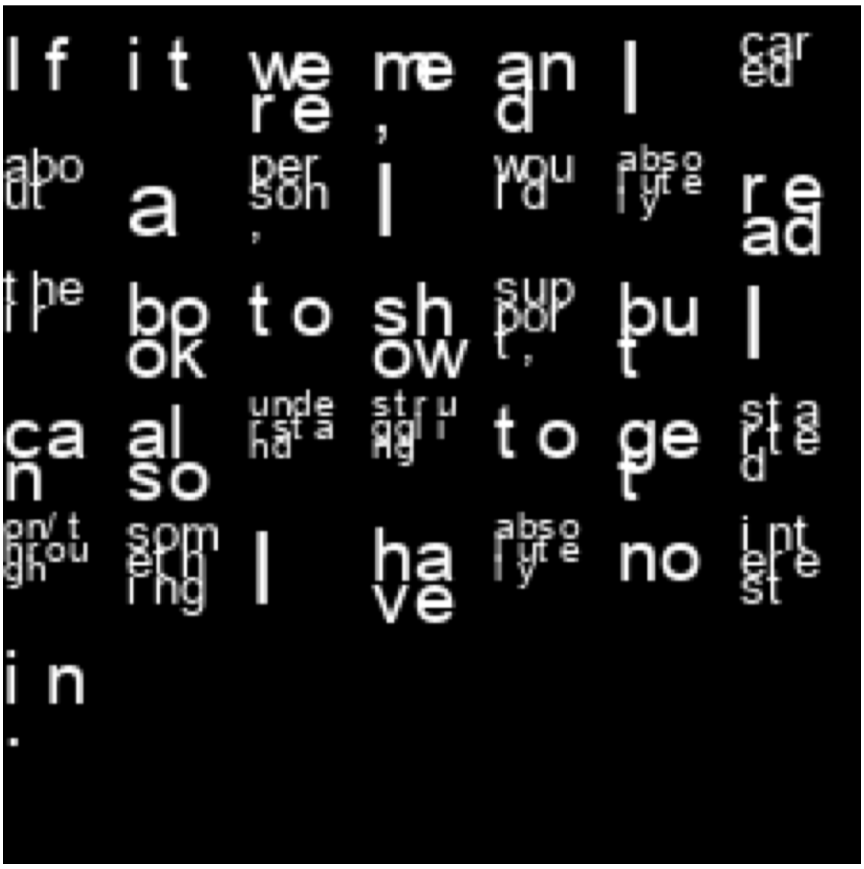


Figure 5.17: An overview of the super-characters model, from the team **GYRFALCON USA**.

Multi-label Text Classification Using an Emotion Embedding Model

Interestingly, the model from team **SKKU SOUTH KOREA** has a similar structure with the model we used in section *MTL with a combination of three embedding methods* 5.5.3. In our work, we tried to combine three embedding methods, *BERT, GloVe and ELMo*, with three different information extraction layers *Conv1D, BiLSTM and BiGRU*, while this team combines two embedding methods, *BERT and GloVe*, with three information extraction layers, but all from the the convolution family: *Conv1, Conv2 and Conv3*, as shown in Fig. 5.18.

The authors of this system description paper did not mention how much does the GloVe embedding improve the model's performance. But the conclusion we reached in our work is that is is better to use only the best embedding method (BERT).

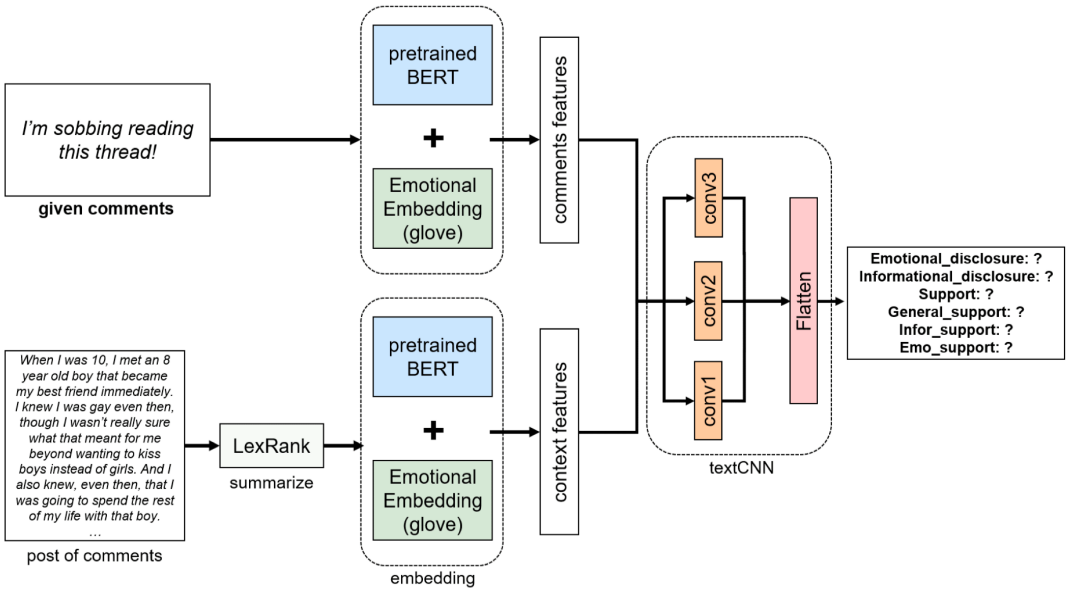


Figure 5.18: Overall System Architecture of Multi-label Text Classification Using an Emotion Embedding Model, from the team **SKKU SOUTH KOREA**.

5.8 Error Analysis

We reviewed the 2,551 wrongly-predicted examples in the test data, from our best model *Venn-diagram-based model* in 5.5.7. For each label, we reviewed the false positive cases and the false negative cases separately, and found some frequently-occurring patterns. We only reviewed four labels: *Emotional_disclosure*, *Information_disclosure*, *Emo_support* and *Info_support*, instead of six because only these four labels are released in the labelled test data.

5.8.1 Label *Emotional_disclosure*

Among the 5,000 test cases, the total number of wrongly-predicted *Emotional_disclosure* is 1,075, including 8,64 false positives and 211 false negatives.

False negatives in *Emotional_disclosure*

Let us first describe the false negative patterns. Among the 211 false negative cases, the most prominent characteristic is that the emotion is expressed by emphasis, adverbs, or without explicit emotion words, i.e., the emotion is hidden in the text. For example:

- *Also, there IS life after this ordeal.*
- *You're not horrible, just human.*
- *Please go to therapy immediately.*
- *Dead by the time I went home at 5pm.*
- *You are not overreacting at all.*

In these cases, the model performs badly and cannot understand the implicit emotions in the text.

False positives in *Emotional_disclosure*

Then for false positives, even though the number of cases is bigger than that of false negatives, we could not find a pattern, but many of the sentences seem ambiguous and difficult to label. For example:

- *Good luck, I hope it goes well!!*
- *I was most definitely trapped too.*

- *You have NO limits on what you can do, and you have a strong, awesome woman standing behind you.*

These sentences are all labelled as *false* for *Emotional_disclosure*. However, they use emphasis or adverbs and seem ambiguous even for humans, so the model finds them difficult to handle too.

We treat the examples above as ambiguous cases, but there are some obvious wrongly labelled ones in the test data. For example:

- *That's a nice sacrifice hahaha.* (labelled as *false*)
- *Glad to hear that police were the worst of it for you.* (labelled as *false*)

Both examples have emotional words: *"hahaha", "glad"*, so we believe they should have been labelled as *true*.

5.8.2 Label *Information_disclosure*

For *Information_disclosure*, total number of wrongly-predicted cases is $1,203$, including 930 false negatives and 273 false positives.

False negatives in *Information_disclosure*

There are three frequent patterns in the false negatives of *Information_disclosure*.

The first one is expressing an option, and sentences always start with the subject I or you. For example:

- *I just want to move on.*
- *I hope you get to move away sometime in the near future!*
- *I wish the same for my grandma.*
- *You're not a bad person.*

The second pattern is presenting a statement. Sometimes it sounds like a catchphrase:

- *Live your life for you.*
- *Summer never comes fast enough.*
- *Breakups hurt and it's going to take time.*

And the last one is that the information is hidden in the text (it is implicit). For example:

- *You deserve to love again.* (hidden info: "You" lost love)
- *I'm so sorry for your loss.* (hidden info: "You" in loss)
- *I will tell you that it's not hopeless.* (hidden info: situation is somewhat in hopeless)

The model performs badly in these three situations. For the last group, they seem ambiguous and we will mention them again in the false positives part.

False positives in *Information_disclosure*

Compared with the three patterns in the false negatives, we only found one pattern in the false positives: a sentence uttered by a person that shows a future action. For example:

- *Yes, that's why I haven't confronted her.*
- *Great, I'll tell her she deserves better!*
- *I highly recommend go visit Iceland, it's great!*

Most of the errors in the false positives follow this format.

We also found an ambiguous sentence:

- *I'm sorry that you're feeling down!* (Labelled as *false*)

This sentence is very similar to the one in false negatives: *I'm so sorry for your loss.* (labelled as *true*), but this time it is labelled as false.

We found a wrongly labelled data:

- *I mean I love hanging out with people.* (Showing the hobby of "I", but labelled as *false*)

5.8.3 Label *Emo_support*

For *Emo_support*, model's performance is much better than for the disclosure group, which can be seen from the number of wrongly-predicted cases. The total number of wrongly-predicted *Emo_support* is 404, including 183 false negatives and 221 false positives. The balance between positive and negative prediction is better than for disclosures, too.

False negatives in *Emo_support*

The common pattern we found in the false negatives of *Emo_support* can be shown as: I ¡verb¿ ¡something¿. For instance:

- *I totally empathize with this.*
- *I hope your cat returns quickly.*
- *I know the hurt is still fresh.*
- *I hope it's filled with bliss.*

They shows some emotional support from the author, but the model may consider it as an action not an emotional expression.

False positives in *Emo_support*

For the false positives, there are two frequent errors made by the model. The first one is that the sentence contains only the emotion, but no support:

- *That was so sweet and unexpected!*
- *I'm glad to hear it from another person as well.*
- *Again, I am very sorry.*

The second is the opposite: the sentence contains only support but no emotion:

- *Girl I am right there with you.*
- *Don't give up hope.*
- *They need you now more than ever.*

These two patterns maybe be caused by the model trying to adapt to both the emotion and support metrics, for the label *Emo_support*. However, when one of the metrics is too high, the label will still be triggered, which could cause the errors.

5.8.4 Label *Info_support*

For the last label, *Info_support*, there are 738 wrongly-predicted cases, including 391 false negatives and 347 false positives.

False negatives in *Info_support*

For the false negative cases, the most frequent pattern is that the supportive information is not prominent in the sentences, therefore the model cannot detect it. For example:

- *It's going to take time.*
- *Please be patient with your grief.*
- *Well done for providing for your family.*
- *Seriously, don't divorce him over this.*

False positives in *Info_support*

In the false positives, some *Emo_support* characteristics are considered as *Info_support* by the model:

- *If you love your husband, cut ties.*
- *Don't let your mind tell you anything else!*
- *Stay strong and keep going.*
- *If you really want to blame anyone, blame the former British Empire.*

5.9 Summary and Directions of Future Work

Our models had the best predictive performance on the Disclosure tasks in the CL-Aff Shared Task 2020, and our system description paper was selected as one of the best system paper in the workshop's shared task.

In this chapter, we presented several multi-task deep learning models. The models have reasonable results on some of the labels, but not all, especially not for *General_support*. The reason is that *General_support* classifies quotes and catchphrases, which have less distinctive features than *Emotional_XXX* or *Information_XXX* while having less positive cases appearing in the dataset.

We compared the results between different MTL structure models on both training and test sets, and found that the Venn-diagram-based MTL structure is the best for the task, which is also proved by comparing our results with results from the other teams. We also found that the performances of two activation functions, LeakyReLU and PRelu, are very similar, at least on this dataset; using statistic information from LIWC does help somewhat, but not as much as manually labelled auxiliary tasks in experiment 1; and combining different embedding methods may be not a good idea.

One possible direction of further work for this task is to make use of large unlabelled data provided. An idea here is to use it to find the different patterns in the texts in order to split them into several groups, and then to train models on each group. Sentences can have different patterns and structures, which require different mapping functions in the network. If we can separate them into clusters in which sentences have similar patterns, there might be an improvement in the classification results.

Chapter 6

Conclusion and Future work

6.1 Conclusion

In this thesis, we investigated applications of multi-task deep learning from two natural language processing tasks.

In Chapter 2, we first introduced the structure of the neural network, including neurons and the backpropagation. Then, we discussed different embedding algorithms and how a neural network can use them to solve natural language problems. After that, we focused on the mechanism of multi-task learning. We discussed the motivation behind them, and how multi-task learning works on noisy data. Finally, we proposed different multi-task learning approaches, including hard parameter sharing and soft parameter sharing.

Two experiments were implemented based on the idea of multi-task deep learning.

The first one in chapter 4 was based on the CL-Aff Shared Task 2019. We presented our multi-task deep learning model for detecting happiness ingredients using two different embedding methods, GloVe and ELMo. Our model works well on the provided happiness text data and obtained acceptable accuracy and F1-score, especially on the label *Social*. Then we focused on comparing the performance difference between single-task learning and multi-task learning, and between different word embeddings. The difference between models shows that multi-task learning model has a stronger ability to represent multi-label tasks over multiple single-task learning models, and using pre-trained trainable GloVe or ELMo with auxiliary tasks can get the best result. Based on the results in 4.6, we have the following conclusions:

- Multi-task learning models have better results than single-task learning models;
- Pre-trained word embeddings can help improve the model performance;
- Allowing the word embeddings update themselves during training can get better results than using fixed word embeddings;
- The auxiliary task helps in the multi-task learning, even though the help may be tiny on some labels;

- The ELMo embedding has a very close but slightly better result over the GloVe embedding, whereas the implementation of ELMo is much more complex.

In chapter 5, we focused on different multi-task deep learning structures. We compared the results between different MTL structure models on both training and test sets, and found that the Venn-diagram-based MTL structure is the best for the task, which is also proved by comparing our results with results from the other teams. The Venn-diagram-based MTL structure model we proposed is a network structure based on Venn diagrams. The original layer is cut into several fragments, and the connections in the neural network are not from layers to layers but from fragments to fragments, based on the Venn diagram. The model obtained the best results in the CL-Aff Shared Task 2020 (for disclosure labels we had the best results, and for supportiveness we had a score very close to the first team's). We also found that the performances of two activation functions, LeakyReLU and PRelu, are very similar on the shared task dataset; using statistic information from LIWC does help somewhat, but not as much as manually labelled auxiliary tasks in experiment 1; and combining different embedding methods may be not a good idea.

6.2 The Limitations of Deep Learning for Classifying Natural Language Content

The limitations of deep learning for classifying natural language content come from two aspects: the limitation of the model, and the limitation of the knowledge representation.

The former one is obvious. Deep learning models are not interpretable; a human cannot understand why the classification decisions were made. Also, the models, no matter how many deep layers they have, will eventually reach a limit on the model performance. They might be surpassed by the transformer architecture, or by some other new models that might appear.

About the latter one, the knowledge representation limits the understanding of the text. Word embeddings are reasonable representations. BERT could be even better. In the future, there might be more powerful word embeddings which can surpass the current ones.

6.3 Summary of the Contributions

We ran extensive experiments to show the superiority of multi-task learning over single-task learning on the CL-Aff shared task 2019 dataset, and we verified the effectiveness of pre-trained embeddings, including GloVe and ELMo, on this dataset [Xin and Inkpen, 2019].

In experiment 2, we compared many multi-task deep learning models. Our contribution here is a novel way to implement hard parameter sharing using Venn diagrams. In the CL-Aff shared task 2020 competition, this model obtained the best score in the disclosure label

group and was very close to the best score in the supportiveness label group. Our paper [Xin and Inkpen, 2020] was chosen as one of the two the best system description papers.

6.4 Future work

Even though we have implemented many models in this thesis, there are many possibile directions of further work.

For example, the embeddings we used in experiment 2 were generated by bert-as-service. Even though bert-as-service is based on BERT, it cannot perform as good as the original BERT perfectly. Due to the memory limitation in our GPU, we could not implement the full BERT in our model. But there is good news: a new embedding method called ALBERT has been published. ALBERT (A lite BERT)[Lan et al., 2020], just as its name implies, is a lite version of the original BERT. ALBERT uses less computing resources, while the performance is as good as BERT's. So, if we can implement ALBERT in our model as an embedding layer in future work, we might obtain better results.

Furthermore, how to learn from the unlabelled dataset is still a problem for both experiments. We did not use the unlabelled in our current models, but we propose to use a bootstrap algorithm: to run our current best model on the unlabelled data, then add to the labelled training data the best of the automatically-labelled instances, namely the ones for which the confidence in the prediction is high for both classes (Social and Agency); then to retrain our model on the enhanced training data. The model trained by this bootstrapping method might work better, but only if we do not add too much noise to the training data.

Another direction of future work for both of our experiments 1 and 2, is the usage of the transformer architecture. Nowadays, transformers have become a hot topic. In the paper *Attention is all you need* [Vaswani et al., 2017], the authors described the mechanism of attention, which then became the module for transformer network. A lot of works have been developed based on this idea. Among them, the most famous one may be the paper *Attention Is (not) All You Need for Commonsense Reasoning*[Klein and Nabi, 2019]. In fact, we tried using the transformer as the classification method in experiment 2, but the results were not as good as we expected. The transformer classifier we used is from a public PyTorch library and it is not specialized for our task, which may be the reason why the performance was not good. In the future, we may consider building a transformer classifier by ourselves from scratch, using the TensorFlow library.

APPENDIX

.1 Python Libraries Used

Besides the built-in Python libraries, the major third-party libraries we used during coding are as follows: *numpy, pandas, sklearn, pytablewriter, tensorflow, tensorflow_hub* and *keras*.

- **Numpy** [Wikipedia, 2020f] is the fundamental package for scientific computing with Python and has high compatibility and integration with other computation libraries, such as tensorflow or keras. The most used feature in Numpy is the array object. Numpy arrays are more efficient and easy to use compared with the built-in list object in Python.
- **Pandas** [Wikipedia, 2020g] is a fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language. Pandas has a huge amount of table processing functions, which is a perfect fit for our experiments because our datasets are all in csv format.
- **Sklearn** [Wikipedia, 2020h] is a machine learning library for Python. It features various classification, regression and clustering algorithms, including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
- **Tensorflow** [Abadi et al., 2015] is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.
- **Tensorflow Hub** is a library for the publication, discovery, and consumption of reusable parts of machine learning models. A module is a self-contained piece of a TensorFlow graph, along with its weights and assets, that can be reused across different tasks in a process known as transfer learning.
- **Keras** [Wikipedia, 2020c] is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

- **pytablewriter** is a Python library to write a table in various formats: CSV / Elasticsearch / HTML / JavaScript / JSON / LaTeX / LDJSON / LTSV / Markdown / MediaWiki / NumPy / Excel / Pandas / Python / reStructuredText / SQLite / TOML / TSV.

.2 How training works

.2.1 Cross-validation (K-Fold)

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

Why cross-validation is useful? In traditional machine learning training, we usually split the dataset into: training, validation and test. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets. And cross-validation is a solution for this problem.

During cross-validation, a test set should still be held out for final evaluation, but the validation set is no longer needed when doing cross-validation. In the basic approach, called k-fold cross-validation, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k "folds":

- A model is trained using $k - 1$ of the folds as training data;
- The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as inverse inference where the number of samples is very small. The whole procedure is shown in Fig. 1.

Because both experiment 1 and experiment 2 are workshop competitions and have separate final test sets, we treat the training data released during the competitions as *Training data* and deploy the k-fold cross-validation, and treat the final test set as the *Test data* in Fig. 1.

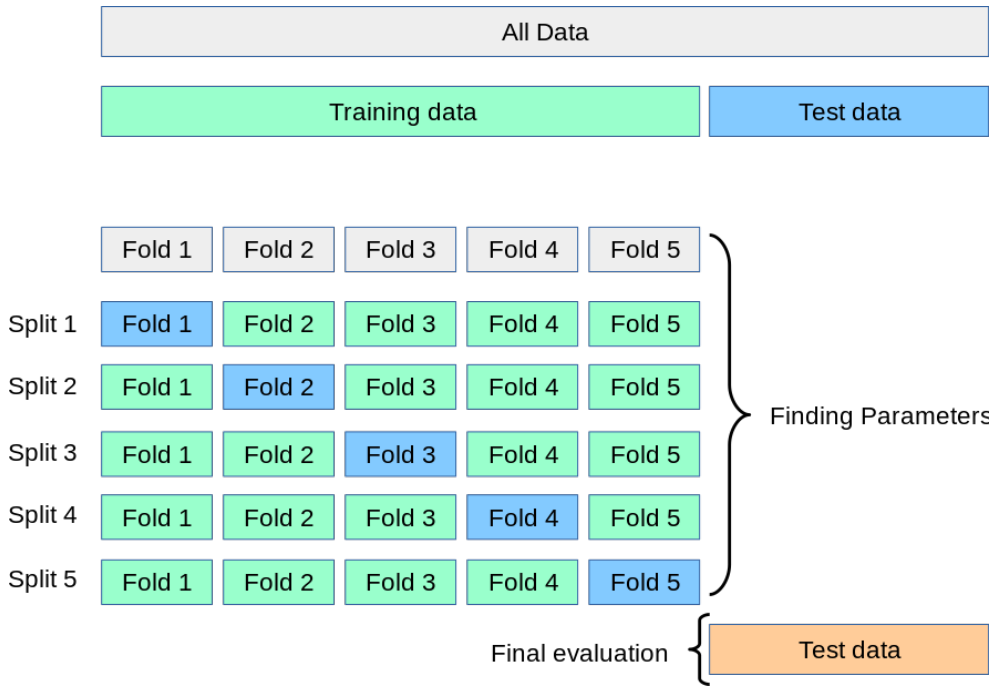


Figure 1: The procedure of a 5-fold cross-validation.

Variations of k-fold

There are two types of cross-validation that can be used in practice(in sklearn): the original K-Fold function and the Stratified K-Fold function.

StratifiedKFold is a variation of KFold. First, StratifiedKFold shuffles your data; after that, it splits the data into n_splits parts. Then, it will use each part as a test set. Note that it only and always shuffles data one time before splitting. Compared with KFold, StratifiedKFold has better performance because it splits the data more balanced according to their labels. But you have to provide the label as the indicator.

Nevertheless, there is another shuffling algorithm, which is very similar to StratifiedKFold, called **StratifiedShuffleSplit**. We know that StratifiedKFold is a variation of KFold, while on the other hand, StratifiedShuffleSplit is a variation of ShuffleSplit. First, StratifiedShuffleSplit shuffles your data, and then it also splits the data into k parts. However, it's not done yet. After this step, StratifiedShuffleSplit picks one part to use as a test set. Then it repeats the same process k - 1 more time, to get k - 1 other test sets. Look at Fig. 2 below, with the same data, but this time, the 4 test sets do not cover all the data, i.e. there are overlaps among test sets.

So, the difference here is that StratifiedKFold just shuffles and splits once; therefore, the test sets do not overlap, while StratifiedShuffleSplit shuffles each time before splitting, and it splits k times, the test sets can overlap.

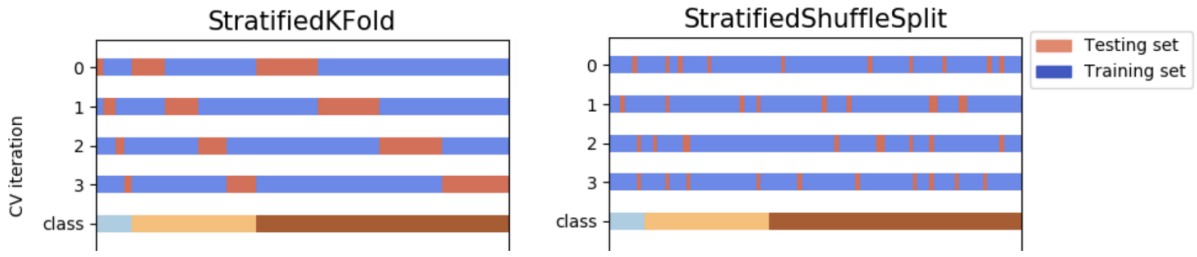


Figure 2: The difference between StratifiedKFold and StratifiedShuffleSplit.

.2.2 About precision, recall, accuracy and F1-score

About the definition of precision, recall, accuracy and F1-score and how to calculate them, please see Fig. 3.

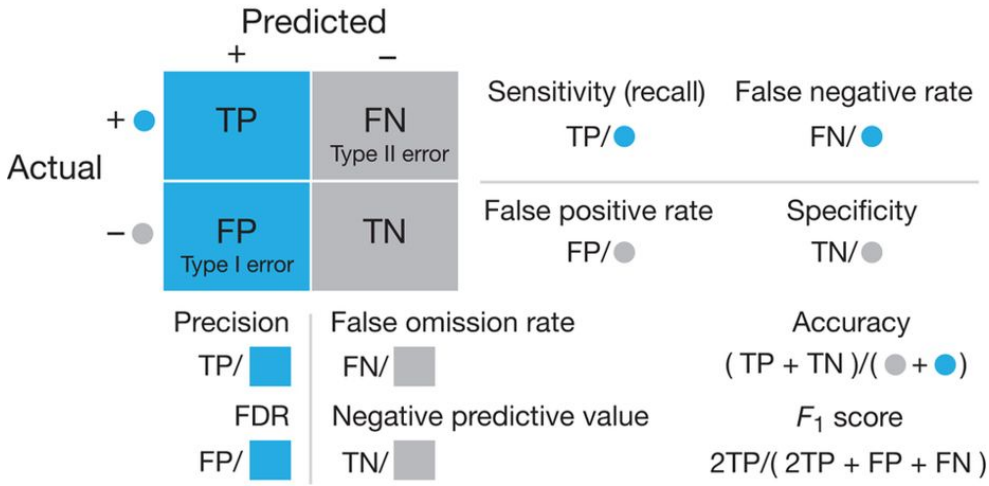


Figure 3: Different metrics and calculation methods.

.3 Usage of ELMo embedding

ELMo embedding [Peters et al., 2018] is a deep contextualized word representation that has great performance among all modern word representations. But training ELMo embeddings from zero will be high time and computation consuming. Therefore, a more efficient way is to use pre-trained ELMo embeddings. As far as I know, there are two official sources providing pre-trained ELMo embeddings.

.3.1 Elmo usage from tf-hub

Tensorflow-hub has implemented ELMo in their library [1] and you can use it as an embedding layer in keras.

The total size of the cache file is 375MB, so it may take a few minutes to download for the first time.

Input

This implementation provides two types of sentence input: when the parameter *signature* is set to **"default"**, ELMo will accept the original full sentences as input; whereas when *signature* is **"tokens"**, users must preprocess the sentences into same-length tokens before feeding them into the ELMo model.

output

Depending on what the parameter *elmo_tag* is set to, the output will be one of the following shapes:

- **word_emb** contains the character-based word representations with shape [batch_size, max_length, 512].
- **elmo** contains the weighted sum of the 3 layers, where the weights are trainable. This tensor has shape [batch_size, max_length, 1024]
- **default** is a fixed mean-pooling of all contextualized word representations with shape [batch_size, 1024].

How to use it in Keras

Here is an implementation of a ELMo embedding layer in Keras:

```
class ElmoEmbeddingLayer(Layer):
    def __init__(self, tag, trainable, **kwargs):
        print('__init__ start')
        self.max_length = 50
        self.trainable=trainable
        self.tag=tag
        super(ElmoEmbeddingLayer, self).__init__(**kwargs)
        print('__init__ fin')

    def build(self, input_shape):
        print('build start')
```

[1] https://tfhub.dev/google/elmo/3

```
        print(self.name)
        self.elmo = hub.Module(module_url, trainable=self.trainable,
            name="{}_module".format(self.name))
        print('hub load fin')
        if self.tag != 'word_emb':
            self.trainable_weights +=
                K.tf.trainable_variables(scope="^{}_module/.*".format(self.name))
        super(ElmoEmbeddingLayer, self).build(input_shape)
        print('build fin')

    def call(self, x, mask=None):
        print('call start')
        print(x.get_shape())
        result = self.elmo(tf.squeeze(K.cast(x, tf.string), axis=1),
                       as_dict=True,
                       signature='default',
                       )[self.tag]
        print('call fin')
        return result

    def compute_mask(self, inputs, mask=None):
        return K.not_equal(inputs, '--PAD--')

    def compute_output_shape(self, input_shape):
        if self.tag == 'default':
            return (input_shape[0], 1024)
        elif self.tag == 'elmo':
            return (input_shape[0], self.max_length, 1024)
        elif self.tag == 'word_emb':
            return (input_shape[0], self.max_length, 512)
```

Masking is a feature in keras layers to support different input/output shapes between layers. However, some layers, like convolutional 1-D or convolutional 2-D layers, requires fixed input shapes, which means a layer that supports masking will not be accepted, even though you know it will only generate a fixed shape. Unfortunately, ELMo embedding layer supports masking, and it cannot be disabled. So if you want to use Con1D or Con2D in the following layers, the ELMo embedding layer should be followed by a NoMasking layer:

```
class NonMasking(Layer):
    def __init__(self, **kwargs):
        self.supports_masking = True
        super(NonMasking, self).__init__(**kwargs)

    def build(self, input_shape):
        input_shape = input_shape
```

```
    def compute_mask(self, input, input_mask=None):
        # do not pass the mask to the next layers
        return None

    def call(self, x, mask=None):
        return x

    def get_output_shape_for(self, input_shape):
        return input_shape
```

.3.2 Pre-trained ELMo Models

Another way to use ELMo is to use pre-trained models from the official website [2]. On the webpage, the authors provide pre-trained ELMo models with different sizes, as shown in Fig. 4.

Example usage of token embedding can be found in **usage_token.py** on the github repo[3].

About the output shape Among all the pre-trained models on the website, let us use $elmo\_2x1024\_128\_2048cnn\_1xhighway\_weights.hdf5$ as an example. The output shape of the embedding from this file will be 256, because ELMo calculated from both directions of a word using LSTM network. $(Left128, Right128) \rightarrow (256)$ will be the final shape of the output. Therefore, the same word from different sentences will not have the same output unless it has totally same left and right context.

For example, in the following two sentences,

- *Pretrained biLMs compute representations useful for NLP tasks.*
- *They give state of the art performance for many tasks.*

For the last word 'task' in two sentences, in the final embedding output, the first 128 values are different, but the following 128 are all the same because they have the same right context (they both have no words on the right).

[2] https://allennlp.org/elmo
[3] https://github.com/chainer/models/blob/master/elmo-chainer/usage_token.py

Model	Link(Weights/Options File)		# Parameters (Millions)	LSTM Hidden Size/Output size	# Highway Layers>	SRL F1	Constituency Parsing F1
Small	weights	options	13.6	1024/128	1	83.62	93.12
Medium	weights	options	28.0	2048/256	1	84.04	93.60
Original	weights	options	93.6	4096/512	2	84.63	93.85
Original (5.5B)	weights	options	93.6	4096/512	2	84.93	94.01

Figure 4: Official pre-trained ELMo models.

References

Xxv. on the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 10(61):168–171, 1880. doi: 10.1080/14786448008626913. URL https://doi.org/10.1080/14786448008626913.

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

M. A. Al-Ajlan and M. Ykhlef. Optimized twitter cyberbullying detection based on deep learning. In *2018 21st Saudi Computer Society National Computer Conference (NCC)*, pages 1–5, April 2018. doi: 10.1109/NCG.2018.8593146.

Akari Asai, Sara Evensen, Behzad Golshan, Alon Halevy, Vivian Li, Andrei Lopatenko, Daniela Stepanov, Yoshihiko Suhara, Wang-Chiew Tan, and Yinzhan Xu. HappyDB: A corpus of 100,000 crowdsourced happy moments. In *Proceedings of LREC 2018*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

Adrian Benton, Margaret Mitchell, and Dirk Hovy. Multi-task learning for mental health using social media text. *CoRR*, abs/1712.03538, 2017. URL http://arxiv.org/abs/1712.03538.

Leon Bottou. Stochastic gradient descent tricks. 7700:430–445, January 2012. URL https://www.microsoft.com/en-us/research/publication/stochastic-gradient-tricks/.

R. Caruana. Multitask learning: A knowledge-based source of inductive bias. *Proceedings of the Tenth International Conference on Machine Learning.*, 1993a.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997. ISSN 1573-0565. doi: 10.1023/A:1007379606734. URL https://doi.org/10.1023/A:1007379606734.

Rich Caruana, Shumeet Baluja, and Tom Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 959–965. MIT Press, 1996. URL http://papers.nips.cc/paper/1081-using-the-future-to-sort-out-the-present-rankprop-and-multitask-learning-for-me pdf.

Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993b.

Y. G. Cheong, Y. Song, and B. C. Bae. [cl-aff shared task] modeling happiness using one-class autoencoders. *In: Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

D. Claeser. Affective content classification using convolutional neural networks. *In: Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011. URL http://arxiv.org/abs/1103.0398.

Glen Coppersmith, Mark Dredze, Craig Harman, and Kristy Hollingshead. From ADHD to SAD: Analyzing the language of mental health on twitter through self-reported diagnoses. In *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*, pages 1–10, Denver, Colorado, June 5 2015. Association for Computational Linguistics. doi: 10.3115/v1/W15-1201. URL https://www.aclweb.org/anthology/W15-1201.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Chollet François. *Deep learning with Python*. Manning Publications Co., 2018. URL www.amazon.com/Deep-Learning-Python-Francois-Chollet/dp/1617294438.

R. K. Gupta, P. Bhattacharya, and Y. Yang. What constitutes happiness? predicting and characterizing the ingredients of happiness using emotion intensity analysis. *In: Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

Ahmed Husseini Orabi, Mahmoud Husseini Orabi, Qianjia Huang, Diana Inkpen, and David Van Bruwaene. Cyber-aggression detection using cross segment-and-concatenate multi-task learning from text. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, pages 159–165, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W18-4419.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Wu J., R. Compton, G. Rakshit, M. Walker, P. Anand, and S. Whittaker. Cruzaffect at affcon 2019 shared task: A feature-rich approach to characterize happiness. *In: Proceedings of the 2 nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

Kokil Jaidka, Saran Mumick, Niyati Chhaya, and Lyle Ungar. The CL-Aff Happiness Shared Task: Results and Key Insights. In *Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019)*, Honolulu, Hawaii, January 2019.

Kokil Jaidka, Iknoor Singh, Lu Jiahui, Niyati Chhaya, and Lyle Ungar. A report of the CL-Aff OffMyChest Shared Task at Affective Content Workshop @ AAAI. In *Proceedings of the 3rd Workshop on Affective Content Analysis @ AAAI (AffCon2020)*, New York, New York, February 2020.

Yoon Kim. Convolutional Neural Networks for Sentence Classification. *ArXiv e-prints*, art. arXiv:1408.5882, August 2014.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Tassilo Klein and Moin Nabi. Attention is (not) all you need for commonsense reasoning. *CoRR*, abs/1905.13497, 2019. URL http://arxiv.org/abs/1905.13497.

Zhen-Zhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942, 2020.

Zachary Chase Lipton, David C. Kale, Charles Elkan, and Randall C. Wetzel. Learning to diagnose with lstm recurrent neural networks. *CoRR*, abs/1511.03677, 2016.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

James Pennebaker, Ryan Boyd, Kayla Jordan, and Kate Blackburn. The development and psychometric properties of liwc2015, 09 2015.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

A. Rajendran, C. Zhang, and M. Abdul-Mageed. Happy together: Learning and understanding appraisal from natural language. *In: Proceedings of the 2 nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*, 2017.

Bernardino Romera-Paredes, Hane Aung, Nadia Bianchi-Berthouze, and Massimiliano Pontil. Multilinear multitask learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1444–1452, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/romera-paredes13.html.

Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. *ArXiv e-prints*, art. arXiv:1706.05098, June 2017.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017. URL http://arxiv.org/abs/1706.05098.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

M. Saxon, S. Bhandari, L. Ruskin, and G. Honda. Word pair convolutional model for happy moment classification. *In: Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

B. Sun, L. Yang, C. Chi, W. Zhang, and M. Lin. [cl-aff shared task] squared english word: A method of generating glyph to use super characters for sentiment analysis. *In: Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

B. Syed, V. Indurthi, K. Shah, M. Gupta, and V. Varma. Ingredients for happiness: Modeling constructs via semi-supervised content driven inductive transfer learning. *In: Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

B. Talafha and M. Al-Ayyoub. Ioh-rcnn: Pursuing the ingredients of happiness using recurrent convolutional neural networks. *In: Proceedings of the 2nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

Ryota Tomioka, Kohei Hayashi, and Hisashi Kashima. On the extension of trace norm to tensors. In *In Proceedings of NIPS Workshop on Tensors, Kernels, and Machine Learning*, 2010.

J. Torres and C. Vaca. Neural semi-supervised learning for short-texts. *In: Proceedings of the 2 nd Workshop on Affective Content Analysis @ AAAI (AffCon2019). Honolulu, Hawaii*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Wikipedia. Confidence interval — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Confidence%20interval&oldid=944137950, 2020a. [Online; accessed 11-March-2020].

Wikipedia. Deep learning — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Deep%20learning&oldid=942561541, 2020b. [Online; accessed 15-March-2020].

Wikipedia. Keras — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Keras&oldid=945994813, 2020c. [Online; accessed 17-March-2020].

Wikipedia. Multi-label classification — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Multi-label%20classification&oldid=956568064, 2020d. [Online; accessed 08-July-2020].

Wikipedia. Multiclass classification — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Multiclass%20classification&oldid=956752346, 2020e. [Online; accessed 08-July-2020].

Wikipedia. NumPy — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=NumPy&oldid=938471739, 2020f. [Online; accessed 17-March-2020].

Wikipedia. Pandas (software) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Pandas%20(software)&oldid=941563449, 2020g. [Online; accessed 17-March-2020].

Wikipedia. Scikit-learn — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Scikit-learn&oldid=934921028, 2020h. [Online; accessed 17-March-2020].

Rui Xia and Zixiang Ding. Emotion-cause pair extraction: A new task to emotion analysis in texts. *CoRR*, abs/1906.01267, 2019. URL http://arxiv.org/abs/1906.01267.

Han Xiao. bert-as-service. https://github.com/hanxiao/bert-as-service, 2018.

Weizhao Xin and Diana Inkpen. [cl-aff shared task] happiness ingredients detection using multi-task deep learning. In *AffCon@AAAI*, 2019.

Weizhao Xin and Diana Inkpen. [cl-aff shared task] detecting disclosure and support via deep multi-task learning. In *AffCon@AAAI*, 2020.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL http://arxiv.org/abs/1505.00853.

Diyi Yang, Zheng Yao, Joseph Seering, and Robert Kraut. The channel matters: Self-disclosure, reciprocity and social support in online cancer support groups. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300261. URL https://doi.org/10.1145/3290605.3300261.

Yongxin Yang and Timothy M. Hospedales. A unified perspective on multi-domain and multi-task learning. *CoRR*, abs/1412.7489, 2014.

Yongxin Yang and Timothy M. Hospedales. Trace norm regularised deep multi-task learning. *CoRR*, abs/1606.04038, 2016. URL http://arxiv.org/abs/1606.04038.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *CoRR*, abs/1707.08114, 2017. URL http://arxiv.org/abs/1707.08114.