Noname manuscript No.
(will be inserted by the editor)

Combination of Loss Functions for Deep Text Classification

Hamideh Hajiabadi · Diego Molla-Aliod · Reza Monsefi · Hadi Sadoghi Yazdi

Received: date / Accepted: date

Abstract Ensemble methods have shown to improve the results of statistical classifiers by combining multiple single learners into a strong one. In this paper, we explore the use of ensemble methods at the level of the objective function of a deep neural network. We propose a novel objective function that is a linear combination of single losses and integrate the proposed objective function into a deep neural network. By doing so, the weights associated with the linear combination of losses are learned by back propagation during the training stage. We study the impact of such an ensemble loss function on the state-of-the-art Convolutional Neural Networks (CNN) for text classification. We show the effectiveness of our approach through comprehensive experiments on text classification. The experimental results demonstrate a significant improvement compared with the conventional state-of-the-art methods in the literature.

Keywords Loss Function · Convolutional Neural Network (CNN) · Ensemble Method · Multi-Class Classifier

1 Introduction

Ensemble methods aim at combining predictions of several estimators in order to obtain better generalization or robustness. It has been shown that an ensemble of single learners can produce a strong learner, and the results produced by ensemble approaches such as random forests and boosting are among the state-of-the-art methods [5, 13].

Loss functions are fundamental components of supervised and unsupervised machine learning systems and are used to train the model parameters. Good loss functions are crucial for the successful training of the model parameters, since standard training methods aim to determine the parameters that minimize the average value of the loss given an annotated training set.

A loss function is a function which provides an indication of error of the estimation during the training stage. There has been much discussion about the characteristics of a good loss function which are briefly explained as follows:-

Convexity The convexity makes optimization easier than the general case since a local minimum must be a global minimum, and first-order conditions are sufficient conditions for optimality. That is why the Square loss function is very popular in the machine learning problems as they tend to lead to a MSE estimator [44, 1].

Hamideh Hajiabadi
Computer Department, Ferdowsi University of Mashhad (FUM)
E-mail: Hamideh.hajiabadi@mail.um.ac.ir

Diego Molla-Aliod
Macquarie University, NSW, 2109, Australia
E-mail: diego.molla-aliod@mq.edu.au

Reza Monsefi (corresponding author)
Computer Department, Ferdowsi University of Mashhad (FUM)
E-mail: monsefi@um.ac.ir

Hadi Sadoghi Yazdi
Computer Department, Ferdowsi University of Mashhad (FUM)
E-mail: h-sadoghi@um.ac.ir

Dual Sparsity Loss functions assign a value to each sample that represents how much that sample contributes to solving the optimization problem. For instance, the Zero-One loss function assigns zero to samples that are correctly classified. In contrast, the Sigmoid loss function gives a non-zero value (possibly very small) to samples that are correctly classified. This means that those samples contribute to solving the optimization problem. Hence, while Zero-one and Sigmoid loss functions are very similar in shape, the Sigmoid loss function leads to a less sparse optimization problem [37].

Margin Enhancing A loss function is margin enhancing if minimization of the expected loss leads to a classifier enhancing the margin [28]. Learning a classifier with an acceptable margin would automatically increase generalization. Enhancing the margin is possible if the loss function returns a small amount of loss for the correct samples that are close to the classification hyperplane. For example, Zero-One does not penalize correct samples at all and therefore it does not enhance the margin, while Hinge Loss is a margin enhancing loss function [28].

Robustness A loss function may lead to a robust classifier if it does not assign high values to outliers. for example, Correntropy loss function, which is shown in Figure 3, has been proven to be a robust loss function [16, 18].

Each loss function comes with its own characteristics, which might make it a good choice for an application and a bad one for the others. In order to make the best use of several loss functions, inspired by ensemble methods for prediction, we propose to use an ensemble of loss functions in the training stage. Through comprehensive experiments, we show how an ensemble loss function improves the performance of a state-of-the-art Convolutional Neural Network (CNN) for text classification. Our contributions are summarized as follows.

1. We apply the ensemble loss function to complex neural networks featuring the state-of-the art CNNs.
2. We study and analyze the contribution of each single loss function towards the ensemble loss.
3. Using the ensemble loss function as a tool, we advance the appropriateness of individual loss functions for various text classification tasks.
4. In Comparison with the current ensemble methods, our approach simultaneously learns the parameter of the learner and the weights associated with every single loss.

The rest of the paper is structured as follows. Section (2) briefly introduces background information and related works. Section (3) introduces the proposed architecture. Section (4) presents the experimental evaluation on several text classification tasks. Finally, Section (5) concludes the paper and discusses future research directions.

2 Related Works

This work concentrates on two broad areas of research: 1) CNNs for text classification, and 2) ensemble methods. In this section, these two areas are covered.

2.1 CNNs for Text Classification

CNNs have been shown to achieve impressive results in the task of text classification [22, 43, 38]. One of the key ideas of CNNs is to apply convolutional layers in order to perform feature extraction and classification as a jointly training task [23]. This scheme has been improved significantly in recent years, especially in the area of computer vision, by using a CNN with many convolution and pooling layers to learn the hierarchical structure of the input [42]. CNN for text classification involves the use of word embeddings for representing words and a CNN aims at detecting the key n-grams features for learning how to discriminate documents on classification problems [2].

Many current Natural Language Processing (NLP) approaches consider words as basic units and introduce words embeddings, which are continuous representations of words [2, 12]. The words in a text document may contain complicated syntactic and semantic relations as a result of local and long dependencies, and consequently it is not clear what is the best representation of a sequence of words. Some approaches model these complex relations by applying Recurrent Neural Networks composed of complex units designed to memorize long-distance dependencies such as Long Short Term Memory (LSTM) [19, 34, 35]. Such approaches, however, are difficult to parallelize and some researchers advocate for the use of CNNs instead. CNNs are usually composed of layers of convolutional filters and pooling filters, which can be implemented as matrix operations that are easy to compute in parallel in Graphical Processing Units (GPUs). The combination of convolutional layers followed by pooling

allows the system to learn to detect the most salient regions of the input space, regardless of the location in the document. In the case of CNNs for text classification, they can detect salient n-grams and can therefore model dependencies spanning several words. CNNs built by a single layer of convolutional filters such as the one proposed by Kim can achieve state-of-the-art (or comparable) results across several datasets [22]. The base architecture that we use in this paper is the CNN proposed by Kim.

2.2 Ensemble Classifiers

Ensemble classifiers are built by a set of individual models, which are named base learners, and combine them to produce improved results [27, 3, 39]. Condorcet's jury theorem was first presented by a French mathematician [9]. This theory studied a jury of voters who need to make a binary decision. Let p be the probability that each voter votes correctly and l be the probability of the majority of voters being correct [9]. Then:

- if $p > 0.5$ then $l > p$,
- if $p > 0.5$ and number of voters tend to infinity then $l \approx 1$.

Although this theorem is a proof of democracy, it can easily be applied to machine learning principles. We therefore expect that by combing many weak learners, which are slightly better than a random guess, a more accurate learner can be produced.

The ensemble supervised learning idea has been in use since the late seventies. In 1977 Tukey proposed combining two regressors. The first regressor fit a line on data and the second one concentrated on remained instances [36]. In 1979 Dasarathy proposed a combined classifier such that each of base classifiers worked on a portion of data [10].

In the nineties, progress in ensemble methods was steady. In 1990 an ensemble of several similarly configured neural networks was proposed. The performance of the combined classifier was improved compared with a single one [17]. The AdaBoost algorithm, introduced in 1996, is one of the most powerful ensemble algorithms. Ensemble algorithms can also be utilized for the purpose of improving the robustness of unsupervised tasks like clustering [40].

Bootstrap AGGregatING (Bagging) was first proposed in 1996 [4]. It improves classification by combining the classification of randomly generated training sets. One of the successful examples of bagging methods is bagging Support Vector Machine (SVM). Since in the SVM algorithm the dimension of the kernel matrix is equal to the sample size, it is better to use a limited sample size [21]. These ensemble methods use several base learners with data which are randomly selected from training data [4].

It is useful to distinguish between dependent and independent ensemble methods. In a dependent ensemble classifier, the output of each base classifier affects learning of the next base classifier. Hence, we can take advantage of the information learned previously for the next classifier. Boosting is an example of a dependent ensemble classifier. Alternatively, in the independent ensemble methods, each base classifier is trained independently and the outputs are then combined for the final verdict. Bagging is an example of an independent ensemble classifier. In this paper, an independent ensemble loss function is applied to a convolutional neural network for text classification.

There has been prior work that shows promising results on the use of an ensemble loss for simple Neural Network (NN) classifiers [15]. The current work shows that the approach can improve the results of the state-of-the-art methods.

3 The Proposed Method

The proposed learning method is shown in Figure 1 with the ensemble loss function represented by the dashed box. Figure 2 shows the CNN architecture of the neural network. This CNN architecture is based on the state-of-the-art CNN for text classification introduced by [22].

3.1 CNN Architecture

Assume that in a multi-class dataset X , $x_i \in \mathcal{R}^K$ is the k-dimensional input that represents the ith word of the sentence. There are C classes such that $y \in \{0, 1\}^C$ is the one-hot encoding of the label. A sequence of

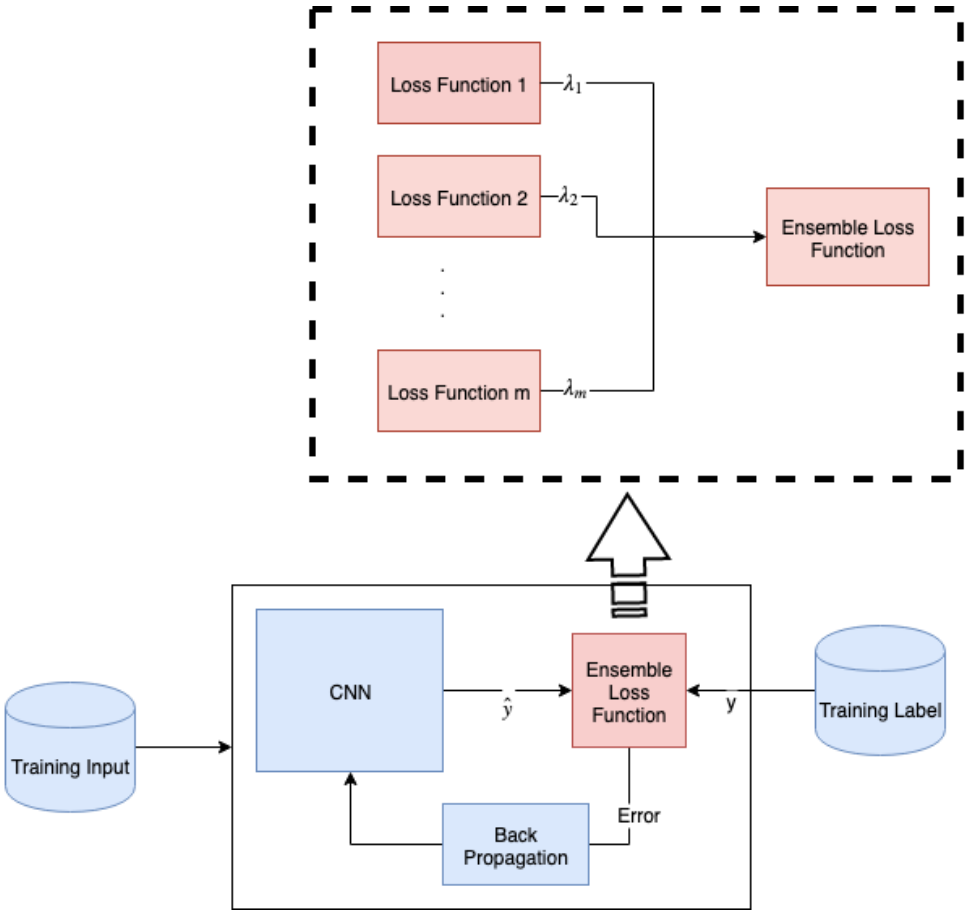

Fig. 1: The Schema of the proposed learning method

length j is represented by concatenating j successive words. Generally we use $x_{i:i+j}$ to represent the sequence $x_i, x_{i+1}, ...x_{i+j}$.

As shown in Figure (1), in a convolutional layer each filter c uses parameter vector $w \in R^{hk}$ which is applied to a sliding window containing h words $x_i, x_{i+1}, ..., x_{i+h-1}$. The same parameter vector w is applied to each window i in the sequence, generating feature c_i as follows:

$$c_i = f(x_{i:i+h-1} \cdot w + b) \tag{1}$$

where $b \in R$ is a bias term and f is a non-linear function. The feature map of each convolutional filter is represented by a vector $[c_1, c_2, ...c_{n-h+1}]$. Then, 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded [8, 7]. It means that the feature with the largest value is the most important one.

In order to consider multiple features, several filter sizes are used. Filters of sizes $3, 4, 5$, are for three-gram, four-gram and five-gram respectively. Each filter results in a univariate feature vector and these are all concatenated to create the input for the next layer. The features, which form the last layer, are passed to a fully connected softmax layer whose output is the probability distribution over labels. Each number in this N dimensional vector represents the probability of a certain class.

By varying the size of the kernels and concatenating their outputs, we are making the opportunity to detect the patterns of multiple sizes (3, 4, or 5 adjacent words). Patterns could be expressions (word n-grams) such as “I hate you”, “very good answer” and so on. CNNs can therefore identify patterns in the sentence regardless of their position. These filters are commonly used for text classification [22] and our work would explore the impact of the ensemble of loss functions on the architecture shown in Figure 2 and their variants.

Figure 2 represents the CNN extended with our ensemble loss function for the 3-class classification problem. A CNN is able to tune its filter weights through backpropagation. In the fully connected layer, a softmax nonlinearity is applied to the output of the network and the ensemble loss function between the normalized predictions and a 1-hot encoding of the label is calculated to train the network’s parameters.

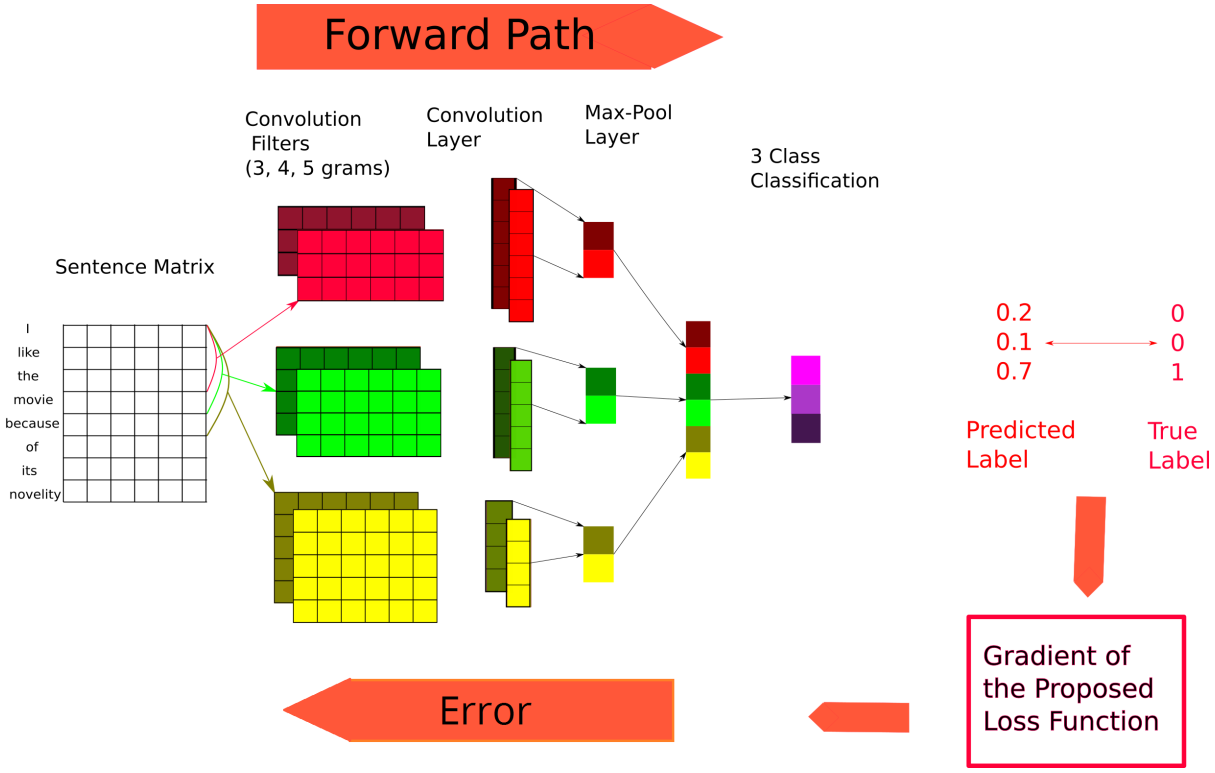

Fig. 2: The proposed CNN for text Classification

In the backward pass, the gradients of our proposed loss function flow back through the networks and the weights contributing most to the loss are determined and are adjusted in order to decrease the loss value. Therefore, the weights are updated in the opposite direction of the gradient.

For regularization, we make use of dropout layers to reduce overfitting. It prevents hidden components from being co-adapted by randomly dropping out a proportion p of the hidden components. Let $z = [\hat{c}_1, \hat{c}_2, ..., \hat{c}_m]$ be the output of the max-pooling layer, the output is generated in the forward path according to Equation (2)

$$y = w \cdot (z \odot r) + b \tag{2}$$

where $r \in R^m$ is a masking vector of Bernoulli variables with the probability p of being 1 and $\odot$ is the element-wise multiplication operator. In the back-propagation pass, the gradients only go through the unmasked components. This helps to overcome over-fitting. We also use the L_2 regularization layer to assure that the network does not over-fit. In the following section, some details about the employed objective function are provided.

3.2 The Novelty of Our Proposed Method

We propose a novel objective function that is a linear combination of single losses and integrate the proposed objective function into a deep neural network. We start with a mathematical explanation of the meaning of a loss function to clarify the importance of the loss function. Let $\hat{y}$ denote the estimated label of a true label y. Then, a loss function $L(y, \hat{y})$ is a positive function which indicates how different $\hat{y}$ is from y. We combine several loss functions using a linear combination with trainable weights. Let $\{L_j(y, \hat{y})\}_{j=1}^{K}$ denote K single loss functions. In addition to finding the optimal parameter of the neural network, the goal is to find the best weights, $\{\lambda_1, \lambda_2, \ldots, \lambda_K\}$, to combine K base loss functions in order to generate a better application-tailored loss function. We need to add a further constraint to avoid yielding near zero values for all λ_i weights. The proposed ensemble loss function is defined as below.

$$L = \sum_{j=1}^{K} \lambda_j L_j(y, \hat{y}), \qquad \sum_{j=1}^{K} \lambda_j = 1 \tag{3}$$

Equation (4) shows the optimization, given N training samples

$$\begin{aligned} \underset{w,\lambda}{\text{minimize}} \sum_{i=1}^{N}\sum_{j=1}^{K}\lambda_j L_j(y_i,\hat{y}_i) \\ \text{s.t.} \sum_{j=1}^{K}\lambda_j = 1, \quad \lambda_j \geq 0 \end{aligned} \tag{4}$$

where y_i are the true labels and $\hat{y_i}$ are the scores. To make the optimization algorithm simpler, we use λ_j^2 instead of λ_j, so the second constraint $\lambda_j \geq 0$ can be omitted. The resulting optimization problem is shown in Equation (5).

$$\begin{aligned} \underset{w,\lambda}{\text{minimize}} \sum_{i=1}^{N}\sum_{j=1}^{K}\lambda_j^2 L_j(y_i,\hat{y}_i) \\ \text{s.t.} \sum_{j=1}^{K}\lambda_j^2 = 1 \end{aligned} \tag{5}$$

We then incorporate the constraint as a regularization term based on the concept of Augmented Lagrangian. The modified objective function using Augmented Lagrangian is presented in Equation (6) [14].

$$\begin{aligned} \underset{w,\lambda}{\text{minimize}} \sum_{i=1}^{N}\sum_{j=1}^{K}\lambda_j^2 L_j(y_i,\hat{y}_i) + \\ \eta_1(\sum_{j=1}^{K}\lambda_j^2 - 1) + \eta_2(\sum_{j=1}^{K}\lambda_j^2 - 1)^2 \end{aligned} \tag{6}$$

Note that the amount of η_2 must be significantly greater that η_1 [31] . The first and the second terms of the objective function cause λ_i^2 values to approach zero but the third term satisfies $\sum_{j=1}^{K}\lambda_j^2 = 1$. Algorithm (1) summarizes the whole training process.

Algorithm 1 Pseudo-Code of the Whole Training Process for the Proposed Method Shown in Figure 2

Input:

The training set T, parameters λ_i (weights associated with each individual loss function), η_1, η_2 (Lagrangian weights) ,σ (Correntopy kernel bandwidth), and m (maximum number of iterations (epochs))

Base loss functions $\{L_i(X_i, y_i)\}_{i=1}^{3}$, K=3 (Correntropy, Multi-class Hinge loss and Cross-Entropy)

Output:

Parameter $W, \lambda_1, \lambda_2, \lambda_3$ in Equation (6)

1: Initiate Ensemble Loss Function using $\{L_j(X_i, y_i)\}_{j=1}^{3}$ and random $\lambda_1, \lambda_2, \lambda_3$
2: Initialize parameters $W \sim N(0, \Sigma)$ and $t = 0$
3: **while** not converged **do**
4: Select a mini-batch of training samples $\{X_i, y_i\}_{i=1}^{N}$ from training set T.
5: Perform a forward path, calculate the loss and regularization term:

$$\sum_{i=1}^{N}\sum_{j=1}^{K=3}\lambda_j^2 L_j(y_i,\hat{y}_i) + \eta_1(\sum_{j=1}^{K=3}\lambda_j^2 - 1) + \eta_2(\sum_{j=1}^{K=3}\lambda_j^2 - 1)^2$$

6: Perform a backward propagation by the BP algorithm.
7: Update $W, \lambda_1, \lambda_2, \lambda_3$ by gradient descent algorithm.
8: $t \leftarrow t + 1$
return $\{\lambda_1^{(t)}, \lambda_2^{(t)}, \lambda_3^{(t)}, W^{(t)}\}$

3.3 Choice of Loss Functions

We combine three loss functions to form our ensemble loss: 1) Multi-class Hinge that enhances the margin, 2) Correntropy that leads to a robust classifier, and 3) Cross-entropy that improves probability estimation. The three chosen loss functions are briefly discussed below.

Multi-Class Hinge loss function. This loss function does not penalize the correctly classified samples that are far away from the classification hyperplane, so, it leads to some (not guaranteed) dual sparsity. However, it does not help with probability estimation that is a big advantage of other loss functions such as the Logistic loss function. Instead, it punishes both the misclassified samples and the ones that are close to classification hyperplane (this is why it is so useful to determine margins). In particular, diminishing hinge-loss comes with diminishing across margin misclassifications. Hence, it leads to a maximum margin classifier that improves the generalization. The Multi-Class Hinge (MC-Hinge) loss function is defined in Equation (7) [30, 41],

$$L_{MC-Hinge} = \sum_{s_j \neq s_{y_i}} \max(0, s_j - s_{y_i} + 1) \tag{7}$$

where s_j is the score of the jth class (incorrect class) for the ith data point and s_{y_i} is the score of y_i class (correct class). Minimizing this multi-class Hinge loss function is equivalent to making the scores of correct labels much larger than those of incorrect labels.

Correntropy loss function. This loss function was first introduced by Liu within the information theory area [25]. The Correntropy loss function is defined in Equation (8),

$$L_{corr}(\hat{y_i}, y_i) = 1 - \exp(-\frac{||y_i - \hat{y_i}||^2}{\sigma^2}) \tag{8}$$

where y_i are the labels, $\hat{y_i}$ are the scores, and σ is the kernel width. Since the base of correntropy is the Gaussian kernel, it is more likely to be robust in the presence of outliers [6]. Figure 3 shows the Correntropy loss function with different values of σ. We use Correntropy loss function with $\sigma = 1.5$.

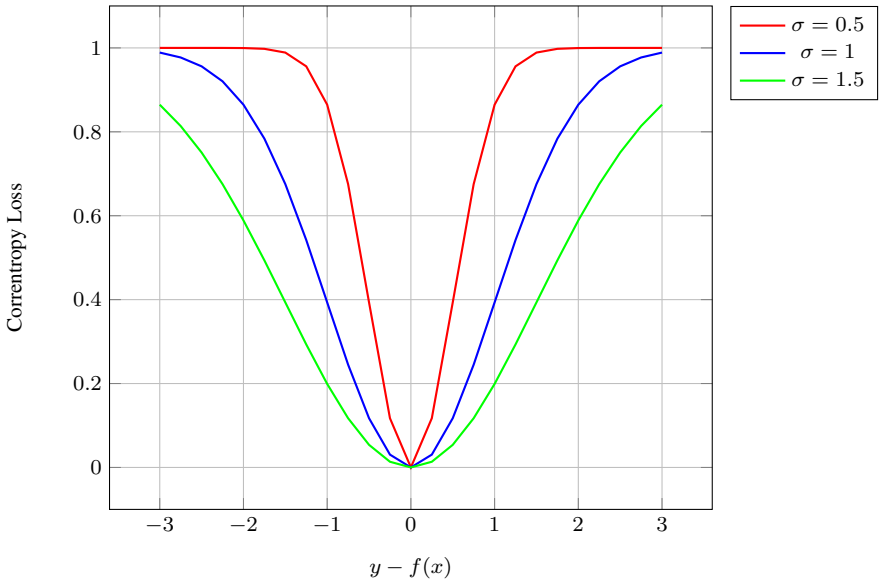

Fig. 3: Correntropy loss function with different values of σ.

Cross-entropy loss function. This loss function is also known as log loss and is the most commonly used loss function in Artificial Neural Networks for classification. The output prediction is always between zero and one and is interpreted as a probability. Data training corresponds to the maximization of the conditional log-likelihood of the data. Cross-entropy loss increases as the predicted probability deviates from the actual label. A perfect model would have a log loss of 0 [11]. Cross-Entropy loss function is defined in Equation (9).

$$L_{c-e}(\hat{y_i}, y_i) = -\sum_i y_i log(\hat{y_i}) \tag{9}$$

4 Datasets and Experimental Setup

To test the applicability of the model for text classification, we have evaluated the proposed model on several benchmark datasets. These are described in Table 1, along with some summary statistics. By testing the model on various text classification tasks on different domains we can explore the general applicability of our model to text classification and generalize over any particular bias or structures that might occur in specific tasks and domains.

4.1 CNN Model

Deep networks have achieved state-of-the-art results in the case of text classification on benchmark datasets. We apply the ensemble loss function to the following variants of the CNN architecture.

CNN-rand: In this model, all the word vectors are randomly initialized and modified during the training process.
CNN-static: In this model, the word vectors are initialized with pre-trained values, using word2vec. The vectors are kept unchanged during the training phase. We utilize the publicly available word2vec vectors of 300 dimensions trained on billions of words from Google News [29].

4.2 Hyperparamteters

The hyperparameters of the network have been set as in Table 2.

Table 1: Description and summary statistics for the datasets. c: Number of classes. l: Average number of tokens per sentence. N: Number of documents. $|V|$: Vocabulary size. Test: Number of documents in the test set (CV means there was no standard train/test split and 10-fold cross-validation was used).

Data	c	l	N	$\|V\|$	Test	Description
MR	2	20	10662	18765	CV	A collection of movie reviews with one sentence per review from the Rotten Tomatoes dataset. It contains positive and negative reviews [32].
SST-1	5	18	11855	17836	6246	An extension of MR but with fine-grained labels (very positive, positive, neutral, negative, very negative) [33]
SST-2	2	19	9613	16185	1821	This is like SST-1 but neutral reviews are removed and it consists of binary labels.
TREC	6	10	5952	9592	500	A collection of questions where the task is to determine the question type (whether the question is about person, location, numerical information, etc.) [24].
MPQA	2	3	10606	6246	CV	The MPQA Opinion Corpus contains news articles from a wide variety of news sources manually annotated for opinions and other private states (i.e., beliefs, emotions, sentiments, speculations, etc.) [20].
Subj	2	23	10000	21323	CV	A collection of sentences where the task is to determine whether a sentence is a subjective or objective [32].
CR	2	19	3775	5340	CV	Customer reviews about products where the task is to determine the polarity of the review (positive or negative) [20].

Table 2: CNN's Configuration

Description	CNN-rand	CNN-Static
Word embeddings	128	300
CNN filter size	3, 4, 5	3, 4, 5
Activation function	ReLu	ReLu
Pooling	1-max pooling	1-max pooling
Dropout rate	0.5	0.5
Mini-Batch size	128	128
Learning rate	1e-3	0.5 e-3
Number of Filters	128	64
l_2-loss	0.35	0.4
number of epochs	50	50

4.3 Results and Discussion

In this section, we investigate the effectiveness of our proposed approach through comprehensive experiments on the benchmark datasets. We first compare its effectiveness in comparison with the state-of-the-art CNNs for text classification. Next, we compare our proposed model against the use of each individual loss function. Last, we explore how much each individual loss function contributes to the final ensemble loss function.

4.3.1 Comparison with the State-of-the-Art

Kim obtained good results for document classification with a single layer CNN [22]. He used differently sized kernels across the filters to make grouping of word representations at different scales [22]. After his work, several researchers have tried to use a deeper word-based network to improve the results in text classification. It was observed that deeper convolutional neural networks did not improve the performance of the text classification.

Table 3 shows the results of our experiments compared with state-of-the-art methods based on Kim's CNN [22] and an extension by Mandelbaum et al. [26]. Mandelbaum et al's network is based on Kim's, with a difference in the embedding layer for representing words. They have implemented a CNN trained on top of the pre-trained word vectors and have proved that improvement in word representation affects the performance of the CNN [26]. For the sake of a consistent comparison with Kim's method, we have not changed the embedding layer. The hyper-parameters have been set as shown in Table 2.

The results listed in the table are as reported in the original papers. We have replicated the work of Kim and observed that our results were very similar. Overall, we can observe that our proposed method outperforms the other methods in five datasets, equals the best in the MPQA dataset, and TREC is the only dataset where it under performs.

4.3.2 Comparison with Individual Loss Functions

In this section, we compare the effectiveness of our approach in comparison with individual loss functions through experiments on the benchmark datasets of Table 1. The CNN-Static architecture has been used and the hyper-parameters of the network have been set as reported in Table 2. 10-fold cross-validation has been used to tune the parameters.

As explained in Section 3, Correntropy, Cross-Entropy and multi-class Hinge loss functions are selected for the base losses to form our ensemble loss function. Table 4 shows the test accuracy obtained by our proposed method in comparison with its components.

Table 3: Test Accuracy of our method in comparison with literature. Best results are highlighted in boldface.

Model	MR	SST1	SST2	Subj	MPQA	TREC	CR
CNN-rand [22]	76.1	45.0	82.7	89.6	83.4	91.2	79.8
CNN-static [22]	81.0	45.5	86.8	93.0	**89.6**	92.8	84.7
CNN-rand [26]	76.4	41	80.2	91.1	-	97.6	-
CNN-static [26]	80.3	48.1	85.4	92.5	-	**98.2**	-
CNN-rand (ours)	77.1	49.2	88.9	90.0	85.6	87.2	81.3
CNN-static (ours)	**81.8**	**57.0**	**93.9**	**93.6**	**89.6**	91.2	**85.6**

In terms of comparison among individual losses, in almost all datasets, Cross-Entropy outperforms the other loss functions (Correntropy and multi-class Hinge). Cross-entropy is a widely used loss function for multi-class classification in neural networks, especially in conjunction with the Sigmoid activation function, causing the predicted output to be interpreted as a probability. Minimizing the Cross-Entropy loss function corresponds to maximizing the conditional log-likelihood of the data. These characteristics make it appropriate for classification purposes and the results show that Cross-entropy is indeed the best when only one loss function is used.

Table 4 also shows that the ensemble of losses improves on the results of each individual loss, including Cross-Entropy, in almost all datasets. This indicates that the other loss functions complement Cross-Entropy and are able to recover some of the errors introduced by Cross-Entropy.

Table 4: Test accuracy with 10-fold cross-validation. Best results are highlighted in boldface

Data	**MR**	**SST1**	**SST2**	**MPQA**	**Subj**	**TREC**	**CR**
Cross-Entropy	79.8	45.0	85.4	88.9	92.8	**92.5**	84.9
Correntropy	78.9	45.0	88.7	81.6	92.4	90.5	84.5
Hinge	81.0	30.6	83.7	70.1	92.4	87.2	84.0
Ours	**81.8**	**57**	**93.9**	**89.6**	**93.6**	91.2	**85.6**

4.3.3 On comparing the CPU time

We have also explored the CPU time for training the CNN architecture shown in Figure 2 with single losses compared with our proposed ensemble loss function. We have run the experiments on a computer with a 3.40 GHz Intel core i7 processor, 32.00 GB RAM, Ubuntu 18.04 operating system, and the programming environment is Pyhton 3.6 with no process running in the background. Table 5 lists the results in seconds. Overall, the running time for the ensemble loss function is slightly higher than the single losses because of its complexity compared with the others, but the increase of time is very low compared with the total running times.

Table 5: The CPU time (in seconds) of our method in Comparison with other methods (5 epochs)

Data	**MR**	**SST1**	**SST2**	**MPQA**	**Subj**	**TREC**	**CR**
Cross-Entropy	134.273	392.723	196.407	79.044	241.576	45.945	68.516
Correntropy	134.852	390.171	197.975	78.848	240.895	46.840	68.112
Hinge	134.185	389.163	196.964	78.538	240.286	45.842	68.089
Ours	135.266	393.722	198.271	79.419	240.919	48.171	68.901

4.3.4 Exploring the Weights Associated with each Loss Function (λ_i)

We have also explored how each of three base loss functions contribute to the ensemble loss function (Figure 4). The actual sum of weights associated with each loss function is not exactly the value 1 because the constraints integrated into Equation (6) are not hard. Therefore, to reach a consistent scale we have normalized all the weights. The normalized values of λ_i are shown in Figure 4. Overall, Cross-Entropy has the largest weights in all datasets, which means that Cross-Entropy contributes the most to form our ensemble loss function. This is expected given the results of Table 4.

Regarding the Correntropy loss function, as shown in Figure 3, Correntropy with $\sigma = 1.5$ (the one we have used in our experiments) does not penalize samples with big errors, so it tends to be a suitable loss function in very noisy environments, especially in the presence of outliers. Unlike other classifiers, CNN itself is a robust classifier; it works by back propagating the gradient of the loss and it is likely to suffer from the vanishing gradient. Therefore, a loss function that highlights the error might work better than Correntropy. This is another reason that the Cross-Entropy loss function gets the largest weight among the others in the ensemble loss function.

5 Conclusions

In this paper, we have proposed the use of an ensemble loss function consisting of a linear combination of single losses and have integrated the ensemble loss into a CNN. The weights associated with the linear combination of losses are learned by backpropagation during the training stage.

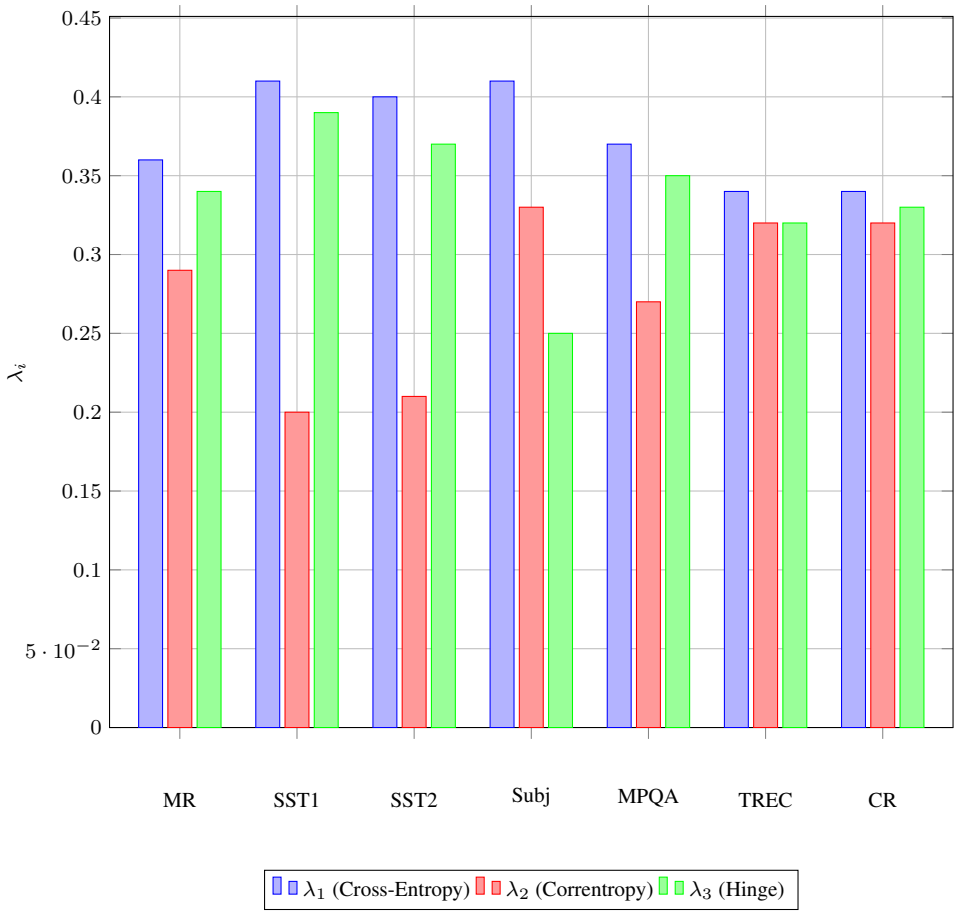

Fig. 4: value of λ_i in different datasets

We have studied how such an ensemble loss function affects the state-of-the-art CNNs for text classification. The proposed loss function showed an improvement compared with the use of the state-of-the-art CNNs in text classification. We have also investigated the effectiveness of the ensemble loss compared with the individual loss functions through experiments. We experimentally showed that an ensemble of loss functions works better than its individual components. Furthermore, associated weights of each individual loss function have been calculated and in almost all datasets, Cross-Entropy obtained the biggest weight.

It is important to note that the only difference between the state-of-the-art CNN architecture and our system is the use of the loss function. This suggests that our proposed loss function may offer improvement over the other CNN architectures. This extension can be added easily and therefore we suggest that it could be added to the currently available neural network architectures. As Table 5 shows, the increase of computation time when adding our proposed loss is rather small.

As a possible next step in our research, we aim at discovering how an ensemble loss function would work in the other applications such as image processing. Another possible extension might be making the weights associated with the combination of losses sparse in order to simplify our model.

References

1. Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.
2. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

3. Luc Devroye Biau, GASrard and GAAbor Lugosi. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, 9(Sep):2015–2033, 2008.
4. Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
5. Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
6. Liangjun Chen, Hua Qu, and Jihong Zhao. Generalized correntropy based deep learning in presence of non-gaussian noises. *Neurocomputing*, 2017.
7. Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.
8. Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
9. Marie Jean Antoine Nicolas Caritat Condorcet. Sketch for a historical picture of the progress of the human mind. 1955.
10. Belur V Dasarathy and Belur V Sheela. A composite classifier system design: Concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713, 1979.
11. Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.
12. Mauro Dragoni and Giulio Petrucci. A fuzzy-based strategy for multi-domain sentiment analysis. *International Journal of Approximate Reasoning*, 93:59–73, 2018.
13. Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Bari, Italy, 1996.
14. Ronald Glowinski and Patrick Le Tallec. *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*, volume 9. SIAM, 1989.
15. Hamideh Hajiabadi, Diego Molla-Aliod, and Reza Monsefi. On extending neural networks with loss ensembles for text classification. *arXiv preprint arXiv:1711.05170*, 2017.
16. Hamideh Hajiabadi, Reza Monsefi, and Hadi Sadoghi Yazdi. relf: robust regression extended with ensemble loss function. *Applied Intelligence*, pages 1–14.
17. Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.
18. Ran He, Wei-Shi Zheng, and Bao-Gang Hu. Maximum correntropy criterion for robust face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1561–1576, 2011.
19. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
20. Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.
21. Hyun-Chul Kim, Shaoning Pang, Hong-Mo Je, Daijin Kim, and Sung-Yang Bang. Support vector machine ensemble with bagging. In *Pattern recognition with support vector machines*, pages 397–408. Springer, 2002.
22. Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.
23. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
24. Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.
25. Weifeng Liu, Puskal P Pokharel, and Jose C Principe. Correntropy: A localized similarity measure. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 4919–4924. IEEE, 2006.
26. Amit Mandelbaum and Adi Shalev. Word embeddings and their use in sentence classification tasks. *arXiv preprint arXiv:1610.08229*, 2016.
27. Shie Mannor and Ron Meir. Weak learners and improved rates of convergence in boosting. In *Advances in Neural Information Processing Systems*, pages 280–286, 2001.
28. Hamed Masnadi-Shirazi and Nuno Vasconcelos. On the design of loss functions for classification: theory, robustness to outliers, and savageboost. In *Advances in neural information processing systems*, pages 1049–1056, 2009.
29. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

30. Robert Moore and John DeNero. L1 and l2 regularization for multiclass hinge loss models. In *Symposium on Machine Learning in Speech and Language Processing*, 2011.
31. Jorge Nocedal and Stephen J Wright. Penalty and augmented lagrangian methods. *Numerical Optimization*, pages 497–528, 2006.
32. Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics, 2005.
33. Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.
34. Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.
35. Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
36. John W Tukey. *Exploratory data analysis*, volume 2. Reading, Mass., 1977.
37. Vladimir Vapnik. *Statistical learning theory. 1998*. Wiley, New York, 1998.
38. Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang, and Hongwei Hao. Semantic clustering and convolutional neural network for short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 352–357, 2015.
39. Wenjia Wang. Some fundamental issues in ensemble methods. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2243–2250. IEEE, 2008.
40. Andreas Weingessel, Evgenia Dimitriadou, and Kurt Hornik. An ensemble method for clustering. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, 2003.
41. Kaobi Yan, Zhixin Li, and Canlong Zhang. A new multi-instance multi-label learning approach for image and text classification. *Multimedia Tools and Applications*, 75(13):7875–7890, 2016.
42. Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
43. Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.
44. Lei Zhao, Musa Mammadov, and John Yearwood. From convex to nonconvex: a loss function analysis for binary classification. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 1281–1288. IEEE, 2010.