PTCN: A Deep Learning Neural Network for Sentiment Classification of Texts

Journal:	*Transactions on Pattern Analysis and Machine Intelligence*
Manuscript ID	Draft
Manuscript Type:	Short
Keywords:	Deep Learning, Convolutional Neural Networks, Long-Short-Term Memory Neural Networks, Natural Language Processing, Sentiment Classification

PTCN: A Deep Learning Neural Network for Sentiment Classification of Texts

Jonathan Wayne Korn
Department of Analytics and Data Science
Harrisburg University of Science and Technology,
Harrisburg, PA
jkorn@harrisburgu.edu*

***Abstract*—This paper aims to expand upon the novel solution's research known as the *"Predict Text Classification Network" (PTCN)*. The PTCN, formerly developed to tackle the binary classification of Congressional Roll Call Votes from legislative texts, is tested on the binary classification of sentiment from texts. The PTCN uses an adaptable hybrid deep learning algorithm which contains the structure of a C-LSTM integrating a convolutional neural network *(CNNs)* and long-short-term memory neural networks *(LSTMs)*. Adaptable parameters in the PTCN's DL architecture allow the network to adjust to various text classification tasks. The PTCN expressed success in accurately predicting the binary classification of texts with varying dimensions (Korn and A.Newman 2020). The PTCN is tested on three samples of texts reflecting different representations of language to further reflect the adaptability of the PTCN to various inputs and generalization to other text classification problems. The first sample is movie scripts, the second is Twitter Airlines tweets, and the third is the IMBD movie reviews dataset. The PTCN average accuracy for each sample is 76.71%, 82.24%, 86.25%, respectively. The adaptable features contained in the PTCN express capabilities to reliably produce accurate results for sentiment classification tasks at the document level compared to past approaches.**

***Index Terms*—Deep Learning (DL); Convolutional Neural Networks (CNNs); Long-Short-Term Memory Neural Networks (LSTMs); Natural Language Processing (NLP); Sentiment Classification**

I. Introduction

Understanding a text's sentiment is a vital component for various interested parties in both academia and industry. For instance, providing insight into a movie script's sentiment could provide hidden knowledge about the user population of a video streaming service. Selecting the right types of movies for consumers is key to businesses such as Netflix and other video content providers. A movie containing mainly positive or negative content can suggest a consumer's opinion towards certain types of sentiment. The recommendation of movies to a user can be altered based on their preferences, providing a measure of sentiment from texts.

Past approaches focused on the formation of the input shapes rather than modifying the model to the task. The bag of words approach was a commonly accepted practice, but for document-level classification tasks, a technique utilizing word vector spaces tends to find issues that arise quickly (Moraes, Valiati, and Neto 2013). The modeling of language is not as simple when more complex representations are introduced into the problem space. Noise is contained in the language no matter the form. Normally the text is very sparse at the document level since a document only contains a small portion of the population's vocabulary (Zhang, Wang, and Liu 2018). Short or long shapes, the texts are going to contain noise that cannot be avoided. Deep models need to adjust to the weights applied throughout the network's layers to learn deeper features throughout the network's credit assignment pathways (CAPs). Not all representations should adjust the model weights significantly unless the input warrants the adaptative weights to scale based on the incoming batch's variant. The PTCN is implemented assuming that most of the noise in language samples is primarily junk data that helps transfer information when humans talk in a socially acceptable manner. In-text classification tasks the importance of transferring information in the same manner as a human is not relevant. The texts are processed to form representations of language with removed noise. The text's sentiment is either positive or negative sentiment. The PTCN adjusts the kernels' initialization and regulation to optimally recognize the best weights to apply to an incoming batch of inputs to significantly alter the model's learning to the predictors' current dimensions. The highly sparse nature of the language utilized within texts is difficult to adjust to; however, the adaptability of the PTCN to the structure of the inputs helps identify latent patterns throughout the samples. The model's adaptable nature tackles high variance in the input's dimensions by allowing adjustment to the weights based on the current inputs while training. Embedding the word vector spaces into the PTCN allows the model to extract features that help infer an accurate decision. Adjusting the convolutional layers to extract features under controlled but adaptable parameters produces an accurate prediction of sentiment from both short and long text samples and is comparable to past approaches.

Promoting a network only to recognize significant features that help class samples of highly sparse texts based on their sentiment provides a valuable advantage in many situations. The simple tokenized word vector spaces are perfect representations of language to feed a deep neural network. The shape represents the purest form of the sampled texts,

containing minimal noise. The set conditions allow the PTCN to capture the most relevant features for the tested sentiment classification tasks.

The experiments express the ability of the PTCN in recognizing sentiment from both short and long text inputs. A corpus of movie scripts reflects the long texts, and the short texts are a corpus of tweets. Being able to perform well on both types of input dimensions expresses the PTCN capabilities to scale it's learning to inputs as they are fed through the network layers.

II. Literature Review

A. Sentiment Classification

Sentiment classification is difficult to address due to the various domains that can be represented in the problem. In 2011, authors introduced a deep learning approach to adapt to sentiment classification tasks that are large-scale (Glorot, Bordes, and Bengio 2011). Expanding the domain representation in training data is difficult to collect, limiting the ability to train for all of them. Using a deep learning approach allows the model to adapt to the change in domains effectively. However, the approach relies on constructing high-level feature representations using the deep learning approach (Glorot, Bordes, and Bengio 2011). The model's focus is not to classify sentiment from the text but rather on input shaping.

By 2013, researchers were attempting document-level sentiment analysis using a bag of words approach. That data in the experiment is not balanced for training, ensuring that the sample contains bias. However, in the study, they showed an artificial neural network (ANN) produced the most accurate results. Using 3000 terms in their bag of words, the model expressed an 86.5% accuracy rate on the movie review dataset (Moraes, Valiati, and Neto 2013). Even with the data imbalance, the ANN showed a stable precision and recall (Moraes, Valiati, and Neto 2013). The length of the movie reviews most likely helped the stability of the model. A deep learning system developed in 2014 addressed the sentiment classification of message-level Tweets known as "Cooooll". The method focuses on the shaping of the inputs into *"... sentiment-specific word embeddings (SSWE) features ..."* (Tang et al. 2014). Note the features are hand-crafted (Tang et al. 2014). Hand-crafting features are a deceptive practice. The features selected to address the task may include more bias than features automatically recognized from a more pure representation of the language. However, distributed representations of both sentences and documents were introduced in 2014 to help mitigate the weakness of the bag of words approach (Le and Mikolov 2014). Most importantly, the research shows that a paragraph vector outperforms a bag of words model.

In 2015, researchers showed that deep CNNs are capable of addressing sentiment analysis problems (Ouyang et al. 2015). The model uses a word2vec representation for the input in their modeling. The team of researchers claims that the vector representation helps initialize good parameters (Ouyang et al. 2015). As well, another team created the UNITN deep CNN targeted for sentiment classification on Twitter texts. The authors stress the importance of initializing the weights of the network (Severyn and Moschitti 2015). To address the issue, the authors implement an unsupervised neural language model to help initialize word embeddings (Severyn and Moschitti 2015). The features recognized in the unsupervised process are used only to initialize the model. In 2015, a team of researchers developed a CNN gated recurrent neural network that outperformed all current methods on the sentiment classification at the document level, reporting an accuracy of 67.6% on the Yelp 2014 dataset (Tang, Qin, and Liu 2015).

In 2016, a team of researchers experimented with a neural network in sentiment classification, and results showed promising performance (Ren et al. 2016). However, the approach is not an effective one for practical purposes. Texts samples often are not simple representations of language such as social media posts but more complete language expressions. The more complex the contextual features, the more bias is likely to develop and become increasingly difficult to combat. The language on social media constantly transforms, making it challenging to adapt a simple neural network to more complex representation texts. One research team reported an accuracy of 46.2% on sentiment classification using a Bidirectional Convolutional Long-Short Term Memory Network (B-CLSTM) (Xu et al. 2016). The B-CLSTM outperformed other neural networks in the experiment. The C-LSTM produced the most comparable accuracy of 42.10% (Xu et al. 2016).

A method of divide and conquer in 2017 is introduced for sentiment classification using a Bidirectional LSTM-CRF and CNN model to adjust to how sentences represent sentiment (Chen et al. 2017). Another discovery in 2017 included a group discovered that utilizing word embeddings with tuning performed better than traditional feature selection methods (Uysal and Murphey 2017). Using finely tuned word embeddings performed at an accuracy of 74.7%, as the model with simple word embedding performed at 71.5% (Uysal and Murphey 2017). The experiment results confirm that a highly tuned feature selection process increases the accuracy of the deep learning model.

Later in 2018, a group of researchers expressed success using a deep neural network for sentiment classification (Ilmania et al. 2018). The approach follows a unique process that preprocesses the input text into a word vector representation, which is utilized to classify sentiment from the input (Ilmania et al. 2018). Assigning a word, sentence, and paragraph to a feature tends to overemphasize a simplification of the text's representation. The inputs' spatial and temporal dimensions must alter throughout the text classifiers' training process to be better generalized.

In 2019, CNN's expressed success in addressing the issue of automatically implementing sentiment classification (Kim and Jeong 2019). Here, the researchers expressed that the CNN performs relatively well on larger texts (Kim and Jeong 2019).

Recently in 2020, a team of researchers introduced a hybrid deep learning model for sentiment classification. The experiment expresses a combination of word embedding techniques and different learning methods, including LSTM, GRU, BiLSTM, and CNN (Salur and Aydin 2020). Interestingly, the features are extracted using the hybrid method using the different deep learning word embedding techniques. The features are utilized to classify the sentiment of the texts (Salur and Aydin 2020). The hybrid approach expresses better results than the prior methods deployed for sentiment classification (Salur and Aydin 2020). The architecture of the hybrid network connects a CNN and BiLSTM network. A comparative study from 2020 helps shed light on the top-performing sentiment analysis methods, which suggests that deep learning techniques are the best approach (Dang, García, and Prieta 2020). On the Tweets Airline dataset for binary sentiment classification, CNN reported 90.37% accuracy, and the recurrent neural network (RNN) reported 90.45% (Dang, García, and Prieta 2020). The CNN and RNN reported 80.06% and 82.82% on the Sentiment 140 for binary sentiment classification (Dang, García, and Prieta 2020). The top-performing methods tested on the IMBD Movie review dataset is the CNN and RNN, reporting 86.07% and 86.68%, respectively.

More recently, a comprehensive study showed that deep learning algorithms performed well on multi-class problems as well (Seo et al. 2020). The study tested multiple types of deep learning algorithms, including CNN, RNN, LSTM, Bidirectional LSTM, Gated Recurrent Units (GRU), and Bidirectional GRU. The models were tested using 12 different datasets. Ten of the datasets are available at ("Amazon Data" n.d.). The Bidirectional LSTM performed the best out of the study (Seo et al. 2020). Even though the experiment shows promising results, the study only performs the testing on word and character level inputs. CNN-BiLSTM with character plus FastText embedding produces accurate results, discovered in 2020 using a Turkish corpus (Salur and Aydin 2020). Labeled the *'M-Hybrid'*, the model performed at an accuracy of 82.14%. Compared to a model performed on the same dataset using another approach, the best research reached an accuracy of 69%; all other experiments were lower performance (Zhang, Zhao, and LeCun 2015) ("Character-Level-Cnn" n.d.) (Kim et al. 2016).

More impressive is the encoder GRU or E-TGRU and the Two-State Gated Recurrent Unit or TGRU, developed by a team of researchers in 2020, reported achieving an 89.37% accuracy using the IMBD dataset (Zulqarnain et al. 2020).

B. Deep Text Classification

The ability to store information over long lags of time using an LSTM expresses successful results; since it can recognize patterns in *"... local space and time ..."* (Hochreiter and Schmidhuber 1997). Researchers say that the model also trained much faster and learned much more than other methods, including *"... real-time recurrent learning, backpropagation through time, recurrent cascade correlation, Elman nets, and neural sequence chunking ..."* (Hochreiter and Schmidhuber 1997).

Often classification solutions require human-designed features (Lai et al. 2015). A recurrent neural network (RNN) shows reliable performance for text classification without the features being created by humans. The automatic recognition of patterns from the word representations and max-pooling layers helped the RNN extract key components leading to specific categories.

Hybrid deep neural networks have shown success using text inputs to solve binary classification tasks. Convolutional and Long-short-term-memory neural networks by Zhou et al., 2015 were developed and showed promising results for hybrid architecture on the classification of text problems. The authors of "A C-LSTM Neural Network for Text Classification" express the capabilities of the network at understanding language (Zhou et al. 2015). The combination of the two types of networks provides the strength of both architectures.

In 2019, a team of researchers took advantage of convolutional recurrent neural networks to tackle text classification tasks (Wang et al. 2019). Their experiments showed that the method of using a C-LSTM achieves better success than other networks. Many of the other networks previously explored were based on only a single network (Wang et al. 2019). Single networks consisted of the following layers: an embedding layer, a convolutional layer, a max-pooling layer, a concatenate layer, an LSTM layer, and an FC layer.

III. Method and Materials

A sample of texts represents significant dimensional space to produce high accuracies for binary sentiment classification. The text samples represent feature space for the set task. The words in the language determine the text sample's sentiment based on a polarity measure calculated on the document level; otherwise, the binary classification of the texts is provided with the dataset.

A supervised machine learning algorithm, a deep neural network known as a C-LSTM, a hybrid neural network that includes convolutional and long-short-term-memory layers, makes it possible to extract the sentiment from the text alone. Applying adaptable parameters and structures within the C-LSTM layers provides an architecture capable of reliably performing text classification tasks. The PTCN or *"Predicted Text Classification Network"* is a reliable model for text classification (Korn and A.Newman 2020). All modeling efforts are captured in the following order:

1) Text Pre-Processing & Sentiment Labeling
2) Hyper-Tuning of the PTCN & Best Parameters PTCN Model 10-fold Cross-validation

A. Text Pre-Processing & Sentiment Labeling

A series of natural language processing techniques are required to supervise the training of the PTCN to classify the text sentiment. The text samples contain noise, which is in the form of stop words, uppercasing, special characters,

NA values, punctuation, numbers, and whitespace (Korn and A.Newman 2020). The removal of noise from the text samples helps mitigate the C-LSTM, recognizing patterns within the texts that create bias predictions. Text samples with mitigated noise are not the only requirement for training the PTCN. Another important factor is to create an equal distribution of the labels contained in the sample data. DL models easily form a bias if the representation of the targets is not controlled.

A classification of each of the text samples' binary sentiment is needed. If necessary, the label of sentiment for each text sample is completed by calculating each word's polarity in a text sample and concluding the overall polarity score. The overall polarity score determines the classification of sentiment for each text based on a conditional threshold of above or below a neutral value of 0. If the polarity is above 0, the classification label is positive for that text sample. If the value is below 0, the classification label is negative. Each polarity measure is converted into a binary label depending on the conditional threshold. Otherwise, the data source provided the labels for the sentiment.

B. Hyper-Tuning of the PTCN and Best Parameters PTCN Model 10-fold Cross-validation

The PTCN is a deep learning algorithm that automatically extracts features from an input vector with adaptable kernel initializers. An additional batch normalization layer and dropout layer are implemented to avoid bias in the model's predictions. Deep neural networks transform inputs over extensive credit assignment pathways *(CAPs)*. The more complicated the dimensions of input, then the more complex the CAPs. The labeled sentiment text samples are balanced into an equal distribution to confirm that the features extracted throughout the text inputs' transformation are only significant. The C-LSTM will be trained on an equal amount of random positive sentiment text samples and random negative sentiment text samples. During the hyper-tuning permutations, the samples are embedded into the network's layers in the form of a tokenized word vectors. The space of the tokenized word vector represents the positive and negative texts that the model will extract features from to form accurate predictions. A list of input samples is tokenized in sequence to form numerical values for each represented word across the texts. Each text is transformed into a numerical sequence of tokenized words in the form of vectors. The newly formed word vector spaces are attached to their designated sentiment label. The lists of input vectors and output scalars require further transformation to embed within the PTCN. The text inputs are padded to make each text the same size. The list of padded inputs is transformed into a matrix. The outputs are converted into categorical variables for each sample transforming from a list of scalars into a matrix of binary responses. The complete sample of text and sentiment labels are split for training, validation, and evaluation datasets during the hyper-tuning sessions. A fixed set of samples is reserved for evaluating the model's hyper-tuning permutations and their corresponding parameters. Once the number of permutations is complete, a set of the best parameters are captured based on the highest evaluation accuracy. The best parameters are stored for training the best model. For the best model, a small variation is required to process the sentiment labeled text samples. Texts processing techniques all remain the same, but the sampling for training and validation sets is changed to incorporate 10-fold cross-validation.

After hyper-tuning, the best model is initiated. The PTCN embeds the pre-processed texts into the model's layers and transforms the inputs through the stacks. The inputs are embedded based on their dimensions, including the max features, embedding dimensions, and max length of the inputs. The inputs are piped through a series of layers in the model. 1-dimensional convolutional layers are set with a series of elements to control various parameters of the convolutions. The input samples are large vectors with high sparsity, requiring that the layers be allowed to explore but in a controlled manner. A series of convolutional layers are stacked and activated using a leaky Relu function to promote the model identifying patterns in the inputs. The model does not overfit because max-pooling layers are implemented at specific points in the model's architecture. Heavy regularization techniques using Lasso and Ridge regression are implemented to mitigate overfitting or underfitting further. The model is designed to pipe the inputs through a series of convolutional layers and then into a long-short-term-memory (LSTM) layer set with various parameter controls and activated with a leaky Relu function. In the last layer, the inputs are piped into a fully connected layer activated using a sigmoid function. The model is compiled using a binary cross-entropy function, optimized with a stochastic gradient descent *(SGD)* function, and reporting results in set metrics.

The model is fit to the training dataset of inputs and outputs through a set number of epochs. Deep learning models validate after each epoch. A specific percentage of the training data is set for validation. To ensure the model does not train further than it can learn, early stopping parameters are utilized to automate a change in the optimizer function's learning rate and end the model's training session if improvement plateaus after a set number of epochs. The best weights are captured and set for the final model state that is saved for evaluation. The evaluation dataset is tested using the fitted model and determines the model's ability to generalize to new inputs. Hyper-tuning the model's parameters will ensure the best possible values are set for final modeling. The final model is trained using 10-fold cross-validation to understand the predictive capabilities of the model fully. To ensure that the model understands the language after the best model is captured and cross-validated, visualization of the embedded features provides insight. The features embedded in the network's layers are visualized to express the words that support or contradict the predicted sentiment classification. Understanding the features helps determine the usefulness of the convolutions recognizing characteristics from the word vector spaces.

The layers throughout the network's stack are essential to

the performance of the PCTN. The sequential deep learning model is constructed with a stack of different layers set with several parameters, including: dropout rate, convolutional hidden nodes, LSTM hidden nodes, L1 regularization rate, L2 regularization rate, batch size, input max length, max features, embedding dimensions, Leaky Relu rate, kernel size, epochs, max-pooling size, learning rate, and validation split.

The first layer in the model is the embedding layer, which embeds the tokenized words into the network. Word vector spaces reflect common practice in natural language processing. The convolutional and LSTM layers in the architecture can extract features automatically from the highly sparse dimensional space of the processed inputs. The next layer the inputs are piped into is a 1-dimensional convolutional layer with only a 4th of the set convolutional hidden nodes. The inputs are representations of long and highly sparse word vector spaces, which makes the model weights critically important. The model requires a method to adjust to the high variance between samples. The control of the kernels will help the model steadily learn from the features extracted from the text.

The use of a variance scaling kernel initializer in the network provides a process *"... that is capable of adapting its scale to the shape of weights."* ("1D Convolution Layer" n.d.). It is important to regularize the kernels when utilizing an adaptive initializer. L1 and L2 regularization help mitigate high fluctuations while batching samples through the layers. L1 is Lasso Regression (LR). It is a penalty term measuring the *"... absolute value of the magnitude of the coefficient."* ("Regularizer" n.d.). L2 is Ridge Regression. A penalty term measuring the *"... squared magnitude of the coefficient."* ("Regularizer" n.d.). The main difference between the two methods is the penalty term. The 1-dimensional convolutional layers are set to inputs padded to the same shape. Strides are set to 1L through the convolutions, which are *"... an integer or list of a single integer, specifying the stride length of the convolution.* [*(R Documentation, 2019)* ("Conv1D Layer" n.d.). The convolutional layer is activated using a Leaky Relu function. The leaky relu function allows for a *"... small gradient when the unit is not active..."*, providing *"... slightly higher flexibility to the model than traditional Relu."* ("LeakyRelu" n.d.). The first convolutional layer extracts lower level features from the inputs due to the decrease in the hidden nodes. The reduction of the number of transformations provides control to the adaptable features initializing the weights. A high kernel size provides the layer with the ability to transform the data at a higher level through a few transformations. The second layer piped to is another 1-dimensional convolutional layer with the same parameters set, except that the number of hidden nodes is set to 32. Increasing the number of hidden nodes provides more transformations extracting higher-level features from the inputs. The second convolutional layer is activated using the leaky Relu function. The next layer in the stack is another convolutional layer set at half the set number of hidden nodes.

Overfitting is mitigated in the convolutional layers by using a max-pooling layer set to 4. Following are two more layers of 1-dimensional convolutional layers set to half the number of set hidden nodes and a 4th of the set hidden nodes. All parameters are set the same as the prior convolutional layers. A second max-pooling layer, batch normalization, and dropout layer are set to mitigate overfitting further. The next layer is an LSTM layer with the set at 32 hidden nodes. Variance Scaling kernel initializers and L1 and L2 kernel regularizers are implemented. The LSTM layer is activated using a Leaky Relu function. A dropout layer set at 0.5 is implemented before the output layer. The output layer is activated using a sigmoid function for binary problems and a softmax function if it is multi-class. The model is compiled using a loss function of binary cross-entropy for binary classification problems or a categorical cross-entropy function multi-classification problems. The stochastic gradient descent (SGD) optimizer is set with a learning rate at a specific value based on the hyper-tuning sessions. The learning rate will be adjusted to different rates during the hyper-tuning sessions to provide the best learning rate for the final model. The models will provide training, validation, and evaluation metrics in the form of accuracy percentages. The results will also provide the loss measure of the training, validation, and evaluation.

In Section IV, the PTCN is tested on a new data sample, the Movie Scripts, and compared to the performance of other deep models aimed at sentiment classification.

IV. Results

The following section details three experiments using the PTCN to predict text sentiment on a binary level, positive and negative, at a document level.

A. Movie Script Sentiment

The dataset utilized in the first experiment using the PTCN contains 8280 movie script samples. Each sample includes the movie title and accompanying movie script text. The movie script text is processed to remove noise. Pre-processing the text includes converting all lettering to lower casing, removing stop words, special characters, NA values, punctuation, numbers, and whitespace from the text. After the text is pre-processed, the sentiment of each movie script needs to be measured. Each document using the pre-processed form is set to capture each individual word's polarity score and then the document's overall polarity measurement. The polarity measures are conditionally converted into binary labels representing positive and negative sentiment.

Since DL models are susceptible to forming bias when trained on data that is not equally distributed across sample data, modeling requires a reduction in the sample size. The initial count of positive and negative sentiment labels is shown in Table I. Unequal distribution of the classes will hinder the PTCN in recognizing the most significant features. Balancing the number of positive samples and negative samples is key to mitigate the PTCN from learning bias features.

TABLE I

Number of Positive and Negative sentiment Labels Movie Scripts

0	3828
1	4452

The new count of positive and negative sentiment labels results in Table II.

TABLE II

Equal Number of Positive and Negative sentiment Labels Movie Scripts

0	3800
1	3800

Now the next process changes for the hyper-tune modeling training sessions and the best model parameter training session. Each of the hyper-tuning sessions and best model parameter sessions differ in the implementation. The hyper-tuning modeling session use no cross-validation and a fixed training, validation, and evaluation split size. For instance, in the experiment, the training samples consisted of 7500 samples, and evaluation consisted of 100 samples. The hyper-tune sessions' training sample is further split into a randomized split of 80% for training and 20% for validation. After, hyper-tuning sessions the best parameters are identified, and the best model is trained using 10-fold cross-validation with randomized samples of 80% for training and 20% for evaluating. The data's training portion is further split into another randomized sampling of an 80% split for training, and 20% split the validation dataset.

The PTCN using 10-fold cross validation averaged 76.711% evaluation accuracy with a standard deviation of 10.27. The best model performed at 83.82% evaluation on fold 5 as depicted below in in Figure 1. The minimum evaluation accuracy is reported at 50% on fold 10. The remaining folds performance measures are depicted in III. The learning curve in the 1 suggests that the parameters are well trained. Notice that the learning curve breaks at one point. The consumption of time is controlled using early stopping mechanisms are implemented in the best model's training parameters, which monitor the validation loss of the model over-time for improvement. If the model's loss does not improve over ten epochs, it does not continue to train and ends its learning at the current state.

The training/validation plots indicate that each follows a similar pattern, which suggests that the problem of overfitting could easily be handled. Even though the evaluation accuracy shows significant variance across the folds, it provides evidence of the algorithm's robustness (Korn and A.Newman 2020). No matter the input of movie script texts in the training, validation, and evaluation datasets, the algorithm performed well.

Comparing the PTCN to other approaches tackling the problem of sentiment classification using text is further

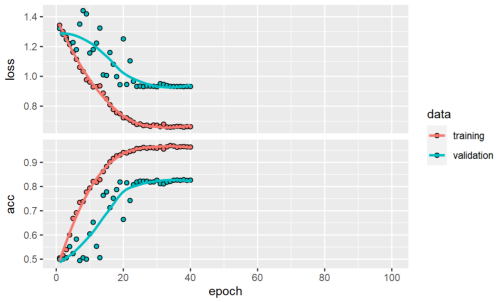

Fig. 1: PTCN performance on fold 3 ˜ Movie Scripts

TABLE III

PTCN evaluation of all folds Movie Scripts

	accuracy	precision	recall	F1
fold1	83.16	83.25	83.03	83.14
fold2	72.76	73.57	71.05	72.29
fold3	82.69	82.74	82.63	82.69
fold4	80.46	80.50	80.39	80.45
fold5	83.82	83.82	83.82	83.82
fold6	82.11	82.19	81.97	82.08
fold7	74.54	74.97	73.68	74.32
fold8	74.54	74.00	75.66	74.82
fold9	83.03	83.03	83.03	83.03
fold10	50.00	50.00	50.00	50.00

explored in two different experiments. The experiments include the binary sentiment classification using the Twitter Airline dataset and IMBD Movie Review dataset. Each dataset is studied by past approaches and provides a benchmark to compare the PTCN's performance.

B. Twitter Airline Sentiment

The Twitter Airline dataset in its original state contains 14640 text samples labeled with a sentiment score. Each sample tweets are pre-processed using the techniques mentioned above discussed in III. The same sentiment process is not performed on the Tweets as the data provided comes with the sentiment classified. Many of the samples are a neutral sentiment score of 0 and are discarded. After pre-processing and ensuring an equal distribution of labels, the sample size decreases to 4720 observations of tweets from various Twitter users representing positive and negative sentiment.

The baseline performance is an RNN at 90.45% and a close second the CNN at a 90.37% accuracy rate (Dang, García, and Prieta 2020). The PTCN's comparable performance is shown in Table IV.

The implementation of the model using 10-fold cross-validation averaged 82.245% evaluation accuracy with a standard deviation of 13.25. The best model performed at 90.89% evaluation on fold 9 as depicted below in in Figure 2. The minimum evaluation accuracy is reported at 52.97% on fold 1. The remaining folds performance measures are depicted in IV. The learning curve in the figure above suggests that the parameters are well trained. The folds do not express significant variance as only fold 1 and fold 10

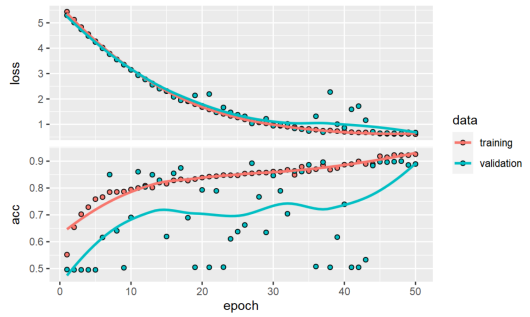

Fig. 2: PTCN performance on fold 9 ˜ Twitter Airline

TABLE IV
PTCN evaluation of all folds Twitter Airline

	accuracy	precision	recall	F1
fold1	52.97	53.02	52.12	52.66
fold2	88.56	88.72	88.35	88.54
fold3	84.22	84.29	84.11	84.20
fold4	88.98	88.82	89.19	89.01
fold5	87.92	88.09	87.71	87.90
fold6	89.72	89.81	89.62	89.71
fold7	87.71	87.55	87.92	87.74
fold8	89.09	89.17	88.98	89.08
fold9	90.89	90.89	90.89	90.89
fold10	62.39	62.53	61.86	62.19

seem to perform at a lower rate. All other folds performed at a stable and higher performance rate. The results suggest that the PTCN performed well at the desired task.

C. IMBD Sentiment

To expand on the comparable performance of the PTCN performing sentiment analysis, the model is trained on the IMBD movie review task. The original dataset contains 50000 observations and the binary labels of sentiment for each text. Pre-processing is the same performed as discussed in III. After, pre-processing the samples are still 50000 due to the fact that the original sample provides an equal representation of sentiment labels. The baseline performance of a model on the IMBD is 89.37% established by (Zulqarnain et al. 2020). The PTCN's comparable performance is shown in Table V.

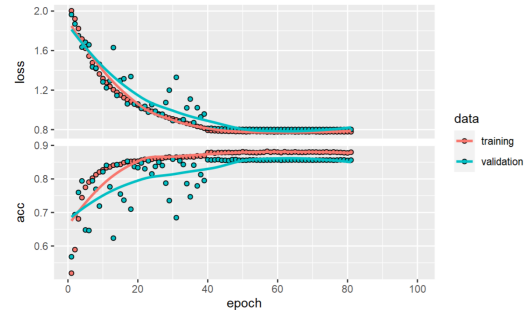

Fig. 3: PTCN performance on fold 2 ˜ IMBD Movie Reviews

The implementation of the model using 10-fold cross-validation averaged 86.254% evaluation accuracy

TABLE V
PTCN evaluation of all folds IMBD Movie Reviews

	accuracy	precision	recall	F1
fold1	85.81	85.89	85.70	85.79
fold2	86.99	86.86	87.16	87.01
fold3	86.34	86.33	86.36	86.34
fold4	86.39	86.35	86.44	86.40
fold5	86.14	86.08	86.22	86.15
fold6	86.47	86.43	86.52	86.48
fold7	86.29	86.33	86.24	86.28
fold8	85.94	85.95	85.92	85.94
fold9	86.60	86.48	86.76	86.62
fold10	85.57	85.56	85.58	85.57

with a standard deviation of 0.41. The best model performed at 86.99% evaluation on fold 2 as depicted below in in Figure 3. The minimum evaluation accuracy is reported at 85.57% on fold 10. The remaining folds performance measures are depicted in IV. The learning curve in Figure 3 suggests that the parameters are well trained. The folds do not express significant variance as only fold 1, fold 8, and fold 10 seem to perform at a lower level. All other folds performed at a stable and higher level. The results suggest that the PTCN performed well at the desired task.

V. Conclusion and Discussion

The PTCN is capable of performing comparably to past approaches in the binary sentiment classification of texts. The unique architecture of the PTCN allows it to mimic adaptable learning. The PTCN expressed high accuracies on the movie script experiment. The PTCN is able to perform comparably to the benchmark models in the twitter Airline and IMBD sentiment experiments. However, the PTCN did slightly perform higher than the benchmark model using the Twitter Airline dataset, but only on one fold and by a very small difference. The IMBD sentiment experiment's benchmark model outperformed the PTCN; however, the PTCN's performance can be improved by introducing new samples.

Instead of maintaining the model's learning in a static state, it is best to set the model to learn optimally from the current inputs the model is processing. It is important to manage the regularization utilized with the PTCN, which is identified through the hyper tuning sessions. The best parameters for the final implementation of the PTCN state are required to produce accurate and reliable results. The folds that express low performance levels are most likley iterations of the model which did not control the exploration of the model successively. It does not seem to be a problem for the successful implmentation of the PTCN.

In the future, the plan is to add an ensemble process into the PTCN and develop the PTCN-2, which will take adavantage of the hyper-tuning sessions and store the various model states to ensemble into a final model state. Using ensembled model state will provide a more robust model that is better at generalizing.

VI. References

"1D Convolution Layer." n.d. Accessed July 25, 2020. https://keras.rstudio.com/reference/layer_conv_1d.html.

"Amazon Data." n.d. Accessed August 1, 2020. http://jmcauley.ucsd.edu/data/amazon/.

"Character-Level-Cnn." n.d. Accessed August 1, 2020. https://github.com/chaitjo/character-level-cnn.

Chen, Tao, Ruifeng Xu, Yulan He, and Xuan Wang. 2017. "Improving Sentiment Analysis via Sentence Type Classification Using Bilstm-Crf and Cnn." *Expert Syst. Appl.* 72: 221–30.

"Conv1D Layer." n.d. Accessed July 25, 2020. https://keras.io/api/layers/convolution_layers/convolution1d/.

Dang, Nhan Cach, María N. Moreno García, and Fernando de la Prieta. 2020. "Sentiment Analysis Based on Deep Learning: A Comparative Study." *ArXiv* abs/2006.03541.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. 2011. "Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach." In *ICML*.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9: 1735–80.

Ilmania, Arfinda, Firman Abdurrahman, Samuel Cahyawijaya, and Ayu Purwarianti. 2018. "Aspect Detection and Sentiment Classification Using Deep Neural Network for Indonesian Aspect-Based Sentiment Analysis." *2018 International Conference on Asian Language Processing (IALP)*, 62–67.

Kim, Hannah, and Young-Seob Jeong. 2019. "Sentiment Classification Using Convolutional Neural Networks." *Applied Sciences* 9: 2347.

Kim, Yoon, Yacine Jernite, David A Sontag, and Alexander M. Rush. 2016. "Character-Aware Neural Language Models." In *AAAI*.

Korn, Jonathan Wayne, and Mark A.Newman. 2020. "A Deep Learning Model to Predict Congressional Roll Call Votes from Legislative Texts." *Machine Learning and Applications: An International Journal (MLAIJ)* 7.

Lai, Siwei, Liheng Xu, Kang Liu, and Jun Zhao. 2015. "Recurrent Convolutional Neural Networks for Text Classification." In *AAAI*.

Le, Quoc V., and Tomas Mikolov. 2014. "Distributed Representations of Sentences and Documents." *ArXiv* abs/1405.4053.

"LeakyRelu." n.d. Accessed July 23, 2020. https://keras.io/api/layers/activation_layers/leaky_relu/.

Moraes, Rodrigo Fracalossi de, João Francisco Valiati, and Wilson P. Gavião Neto. 2013. "Document-Level Sentiment Classification: An Empirical Comparison Between Svm and Ann." *Expert Syst. Appl.* 40: 621–33.

Ouyang, Xi, Pan Zhou, Cheng Hua Li, and Lijun Liu. 2015. "Sentiment Analysis Using Convolutional Neural Network." *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 2359–64.

"Regularizer." n.d. Accessed July 23, 2020. https://keras.io/api/layers/regularizers/.

Ren, Yafeng, Yue Zhang, Meishan Zhang, and Donghong Ji. 2016. "Context-Sensitive Twitter Sentiment Classification Using Neural Network." In *AAAI*.

Salur, Mehmet Umut, and Ilhan Aydin. 2020. "A Novel Hybrid Deep Learning Model for Sentiment Classification." *IEEE Access* 8: 58080–93.

Seo, Seungwan, CzangYeob Kim, Haedong Kim, Kyounghyun Mo, and Pilsung Kang. 2020. "Comparative Study of Deep Learning-Based Sentiment Classification." *IEEE Access* 8: 6861–75.

Severyn, Aliaksei, and Alessandro Moschitti. 2015. "UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification." In *SemEval@NAACL-Hlt*.

Tang, Duyu, Bing Qin, and Ting Liu. 2015. "Document Modeling with Gated Recurrent Neural Network for Sentiment Classification." In *EMNLP*.

Tang, Duyu, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou. 2014. "Coooolll: A Deep Learning System for Twitter Sentiment Classification." In *SemEval@COLING*.

Uysal, Alper Kursat, and Yi Lu Murphey. 2017. "Sentiment Classification: Feature Selection Based Approaches Versus Deep Learning." *2017 IEEE International Conference on Computer and Information Technology (CIT)*, 23–30.

Wang, Ruishuang, Zhenwei Li, Jian Cao, Tong Chen, and Lei Wang. 2019. "Convolutional Recurrent Neural Networks for Text Classification." *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–6.

Xu, Jiacheng, Danlu Chen, Xipeng Qiu, and Xuanjing Huang. 2016. "Cached Long Short-Term Memory Neural Networks for Document-Level Sentiment Classification." In *EMNLP*.

Zhang, Lei Johnny, Shuai Wang, and Bing Liu. 2018. "Deep Learning for Sentiment Analysis : A Survey." *ArXiv* abs/1801.07883.

Zhang, Xiang, Junbo Jake Zhao, and Yann LeCun. 2015. "Character-Level Convolutional Networks for Text Classification." In *NIPS*.

Zhou, Chunting, Chonglin Sun, Zhiyuan Liu, and Francis Lau. 2015. "A c-Lstm Neural Network for Text Classification." *arXiv Preprint arXiv:1511.08630*. https://arxiv.org/abs/1511.08630.

Zulqarnain, Muhammad, Suhaimi Abd Ishak, Rozaida Ghazali, Nazri Mohd Nawi, Muhammad Aamir, and Yana Mazwin Mohmad Hassim. 2020. "An Improved Deep Learning Approach Based on Variant Two-State Gated Recurrent Unit and Word Embeddings for Sentiment Classification." *International Journal of Advanced Computer Science and Applications* 11.