Phuoc Le, Gil Rabara, Dilnoza Saidova, Vivian Tran
TCSS 456: Natural Language Processing
May 14, 2023

Deliverable 1 - Model

Source Code Files

Source code has been written and developed in the Google Colab; it is uploaded to the group project Drive as a zip file.

Description & Details of the Model

Architecture of the Neural Network - Convolutional Neural Network (CNN)

CNN, or Convolutional Neural Network, is a type of neural network commonly used in computer vision tasks, such as image classification, object detection, and segmentation. The key feature of a CNN is that it uses convolutional layers, which are designed to automatically learn and detect local features or patterns in an input image.

In the toxic email classification project, a CNN is selected as the neural network architecture for a few reasons. Firstly, an email's text can be considered a 2D image with words and their sequence represented along one axis and the different emails represented along the other axis. By using a 2D convolutional layer, the model can learn patterns or features in the text, such as character and word-level combinations and sentence-level structures. Additionally, CNNs are good at learning relevant features automatically, which is useful in natural language processing tasks where feature engineering can be time-consuming and complicated. By using convolutional layers, the model can learn both low-level and high-level features from the text, improving the accuracy of the classification task.

Overall, a CNN is a powerful tool for the given project due to its ability to learn and detect local features in the text, and its ability to automatically learn relevant features without the need for extensive feature engineering.

Steps Taken to Implement a CNN for Toxic Email Classification

1. **Data Preprocessing Steps:**

    a. **Cleaning**: Removing any unwanted characters, such as special characters, HTML tags, and punctuation, from the text data.
    b. **Tokenization**: Splitting the text data into individual tokens or words, which is used as inputs for machine learning models.
    c. **Stop word removal**: Removing common words such as "the," "a," and "an," which do not provide much meaning to the text.
    d. **Vectorization**: Representing the text data as numerical vectors used as inputs for machine learning models.

e. **Splitting**: Splitting the dataset into training, validation, and test sets, which is used to train, tune, and evaluate machine learning models.

2. **Data Splitting:** The preprocessed dataset is split into training, validation, and testing sets.

3. **Text Encoding:** The text data in the preprocessed dataset is encoded into numerical representations that is fed into the CNN model. One common method is to use word embeddings such as GloVe to represent each word as a dense vector. This step is not used in initial model, and will be added in the deliverable.

4. **Model Architecture:** The CNN architecture for toxic email classification consists of several convolutional layers followed by max-pooling layers to extract local features from the text data. The output of the convolutional layers is then flattened and fed into a fully connected layer with a sigmoid activation function to output a binary classification for each email. The model uses the binary cross-entropy loss funcation and the Adam optimizer with a default learning rate of 0.001. The input sequences are preprocessed by lowercasing, removing punctuation, stemming, and removing special characters before being passed through the model. The below architecture, used for the model development, is a commonly used and effective architecture for text classification tasks:

    a. Input layer takes an input of shape (300,), which is hte maximum length of the padded sequences.
    b. Embedding layer maps each owrd index to a dense vector of size 300.
    c. 1D Convolutional Layer with kernel size 3 and 5 filters with a ‘tanh’ activation function adn ‘same’ padding.
    d. Dropout layer with 0.5 rate.
    e. Bidirectional LSTM layer with 200 unit in each direction, dropout rate of 0.5, adn recurrent dropout rate of 0.25.
    f. Concatentation of the GlobalAverage Pooling 1D and GlobalMaxPooling1D layers.
    g. Fully connected output layer with 6 units and sigmoid activation funcation.

5. **Model Compilation and Training:** The CNN model is compiled with an appropriate loss function (binary cross-entropy), an optimizer (Adam), and a metric (accuracy). The model is then be trained on the preprocessed and split dataset using the fit() function in Keras with the appropriate batch size and number of epochs. Below is code snippet of model training:

```
# Build and train model
inp = Input(shape=(300,))
x = Embedding(5000, 300, trainable=True)(inp)
x = Conv1D(kernel_size=3, filters=5, padding='same', activation='tanh', strides=1)(x)
x = Dropout(0.5)(x)
x = Bidirectional(LSTM(200, return_sequences=True, dropout=0.5, recurrent_dropout=0.25))(x)
x = concatenate([GlobalAveragePooling1D()(x), GlobalMaxPooling1D()(x)])
out = Dense(6, activation='sigmoid')(x)
model = Model(inp, out)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

6. **Model Evaluation:** Once the model is trained, it is evaluated on the testing set using appropriate evaluation metrics such as loss, validation, and accuracy. Later deliverables will use precision, recall, and F1-score.

Evaluation Results

Learning Rate

- This is the step size for updating model parameters during training. It controls how much the weights are updated in each iteration of training. A high learning rate may result in unstable training, while a low learning rate may result in slow convergence.

- For the purpose of simplicity of this project, the learning rate is not explicitly set; therefore, it will use hte default learning rate of the optimizer - Adam with a default learning rate of 0.001.

Batch Size

- This is the number of samples used in each iteration of training. A larger batch size may result in faster training but may require more memory, while a smaller batch size may require more iterations to converge but may be more stable.

- In this project, the batch size is set to 256, as shown in the `model.fit()` function call:
  `model.fit(X_training, y, batch_size=256, epochs=1, validation_split=0.4`

- This means that the model will train on batches of 256 samples ata a time before updating hte model's weights based on the loss computed on that batch.

Number of Epochs

- This is the number of times the entire dataset is iterated over during training. Increasing the number of epochs may result in better performance but may also increase the risk of overfitting.

- The number of epochs in the model is set to 1, as shown in the `model.fit()` function:
  `model.fit(X_training, y, batch_size=256, epochs=1, validation_split=0.4`

- This means that hte model will only train for one epoch over the entire training dataset; it will go through all the training data exactly once.

Regularization

- Dropout is a form of regularization that randomly sets a fraction of the input units to 0 at each update during training time, which helps to prevent overfitting by reducing the sensitivity of the model to the specific weights of individual neurons.

- The model uses dropout regularization, which is applied to the output of the convolutional layer with a dropout rate of 0.5. It is also applied to the output of the LSTM layer with the same rate of 0.5.

- Specifically, in the following code snippet, dropout is applied twice: first after the convolutional layer, and then after the LSTM layer.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(64, input_dim=100, activation='relu'))
model.add(Dropout(0.5)) # dropout layer with rate 0.5
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5)) # dropout layer with rate 0.5
model.add(Dense(1, activation='sigmoid'))
```

- The dropout rate is set to 0.5 in both cases, which means that 50% of the inputs to these layers are randomly set to zero during each training iteration.

Sample Model Training Results with Different Filter Sizes

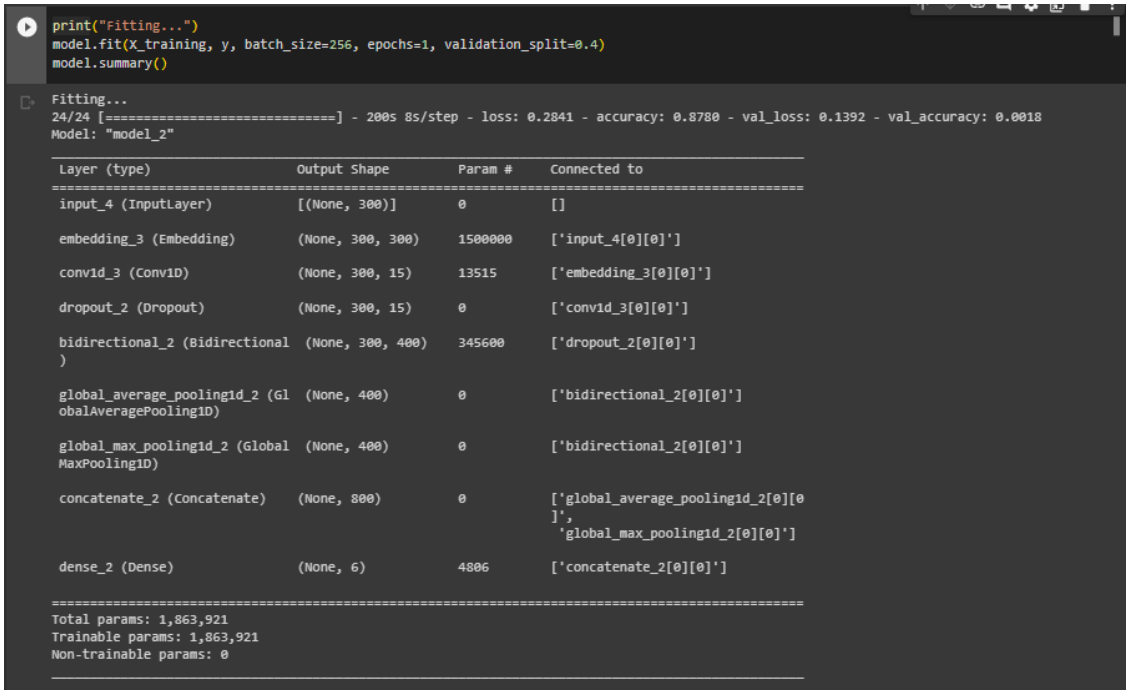
Filter: 5

```
[15] print("Fitting...")
     model.fit(X_training, y, batch_size=256, epochs=1, validation_split=0.4)
     model.summary()

Fitting...
24/24 [==============================] - 196s 8s/step - loss: 0.2726 - accuracy: 0.7038 - val_loss: 0.1387 - val_accuracy: 0.0030
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_2 (InputLayer)        [(None, 300)]             0           []

 embedding_1 (Embedding)     (None, 300, 300)          1500000     ['input_2[0][0]']

 conv1d_1 (Conv1D)           (None, 300, 5)            4505        ['embedding_1[0][0]']

 dropout (Dropout)           (None, 300, 5)            0           ['conv1d_1[0][0]']

 bidirectional (Bidirectional)  (None, 300, 400)       329600      ['dropout[0][0]']

 global_average_pooling1d (Glob  (None, 400)           0           ['bidirectional[0][0]']
 alAveragePooling1D)

 global_max_pooling1d (GlobalMa  (None, 400)           0           ['bidirectional[0][0]']
 xPooling1D)

 concatenate (Concatenate)   (None, 800)               0           ['global_average_pooling1d[0][0]'
                                                                  , 'global_max_pooling1d[0][0]']

 dense (Dense)               (None, 6)                 4806        ['concatenate[0][0]']

=================================================================================================
Total params: 1,838,911
Trainable params: 1,838,911
Non-trainable params: 0
_________________________________________________________________________________________________
```

Filter: 10

```
     print("Fitting...")
     model.fit(X_training, y, batch_size=256, epochs=1, validation_split=0.4)
     model.summary()

Fitting...
24/24 [==============================] - 212s 9s/step - loss: 0.2796 - accuracy: 0.4686 - val_loss: 0.1375 - val_accuracy: 0.9950
Model: "model_1"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_3 (InputLayer)        [(None, 300)]             0           []

 embedding_2 (Embedding)     (None, 300, 300)          1500000     ['input_3[0][0]']

 conv1d_2 (Conv1D)           (None, 300, 10)           9010        ['embedding_2[0][0]']

 dropout_1 (Dropout)         (None, 300, 10)           0           ['conv1d_2[0][0]']

 bidirectional_1 (Bidirectional  (None, 300, 400)      337600      ['dropout_1[0][0]']
 )

 global_average_pooling1d_1 (Gl  (None, 400)           0           ['bidirectional_1[0][0]']
 obalAveragePooling1D)

 global_max_pooling1d_1 (Global  (None, 400)           0           ['bidirectional_1[0][0]']
 MaxPooling1D)

 concatenate_1 (Concatenate)  (None, 800)              0           ['global_average_pooling1d_1[0][0
                                                                  ]',
                                                                   'global_max_pooling1d_1[0][0]']

 dense_1 (Dense)             (None, 6)                 4806        ['concatenate_1[0][0]']

=================================================================================================
Total params: 1,851,416
Trainable params: 1,851,416
Non-trainable params: 0
_________________________________________________________________________________________________
```

Filter: 15



```
print("Fitting...")
model.fit(X_training, y, batch_size=256, epochs=1, validation_split=0.4)
model.summary()
```

```
Fitting...
24/24 [==============================] - 200s 8s/step - loss: 0.2841 - accuracy: 0.8780 - val_loss: 0.1392 - val_accuracy: 0.0018
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 300)]        0           []

 embedding_3 (Embedding)        (None, 300, 300)     1500000     ['input_4[0][0]']

 conv1d_3 (Conv1D)              (None, 300, 15)      13515       ['embedding_3[0][0]']

 dropout_2 (Dropout)            (None, 300, 15)      0           ['conv1d_3[0][0]']

 bidirectional_2 (Bidirectional  (None, 300, 400)    345600      ['dropout_2[0][0]']
 )

 global_average_pooling1d_2 (Gl  (None, 400)         0           ['bidirectional_2[0][0]']
 obalAveragePooling1D)

 global_max_pooling1d_2 (Global  (None, 400)         0           ['bidirectional_2[0][0]']
 MaxPooling1D)

 concatenate_2 (Concatenate)    (None, 800)          0           ['global_average_pooling1d_2[0][0
                                                                 ]',
                                                                  'global_max_pooling1d_2[0][0]']

 dense_2 (Dense)                (None, 6)            4806        ['concatenate_2[0][0]']

==================================================================================================
Total params: 1,863,921
Trainable params: 1,863,921
Non-trainable params: 0
__________________________________________________________________________________________________
```

Resources

Georgakopoulos, S. V., Vrahatis, A. G., Tasoulis, S. K., & Plagianakos, V. P. (2018, February 28). *Convolutional Neural Networks for Toxic Comment Classification*. https://arxiv.org/pdf/1802.09957.pdf

Jigsaw/Conversation AI. (2018). *Toxic Comment Classification Challenge*. Kaggle. https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/overview

Sheffield, Z. Z. U. of, Zhao, Z., Sheffield, U. of, Profile, U. of S., Sheffield, Z. Z. U. of, Zhang, Z., Sheffield, F. H. U. of, Hopfgartner, F., Stanford, Institute, J. S., Google, University, T., Foundation, W., & Metrics, O. M. A. (2021, April 1). *A comparative study of using pre-trained language models for toxic comment classification: Companion Proceedings of the web conference 2021*. Association for Computing Machinery. https://dl.acm.org/doi/abs/10.1145/3442442.3452313

Spiros V. Georgakopoulos Department of Computer Science and Biomedical Informatics, Georgakopoulos, S. V., Department of Computer Science and Biomedical Informatics, Sotiris K. Tasoulis Department of Computer Science and Biomedical Informatics, Tasoulis, S. K., Aristidis G. Vrahatis Department of Computer Science and Biomedical Informatics, Vrahatis, A. G., Vassilis P. Plagianakos Department of Computer Science and Biomedical Informatics, Plagianakos, V. P., & Metrics, O. M. A. (2018, July 1). *Convolutional Neural Networks for toxic comment classification: Proceedings of the 10th Hellenic Conference on*

Artificial Intelligence. Association for Computing Machinery. https://dl.acm.org/doi/abs/10.1145/3200947.3208069

GitHub Links - Other Projects Used for Code References:

https://github.com/arunarn2/ToxicCommentChallenge
https://github.com/vedantapawar/ToxicCommentClassificationNLP