News Headline Classification

Introduction

Due to the development of technology, the dissemination and collection of information have become very easy, especially for news stories of all kinds. Now every news media has its website, and we can easily collect all kinds of news articles from these websites. Even ordinary people can write their news articles on some public websites. When faced with these large numbers of news articles, quickly categorizing them and finding the articles we need becomes particularly important. Since most essays have a high amount of text, it is time-consuming and resource-intensive to analyze the entire article directly. Therefore, people turn their attention from the article as a whole to the introduction of the article. But short text classification is a challenging task because there are several words in each text, usually less than 20 words, which brings about the problem of feature sparseness. We want to use a neural network to create an NLP model that can achieve excellent performance on the news headline classification task.

Dataset

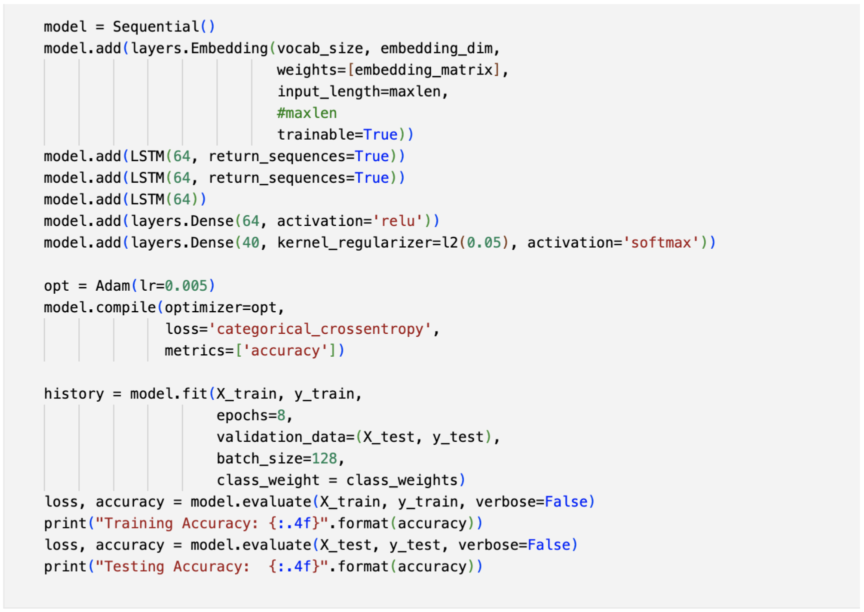
The first dataset we are using is coming from the kaggle. It has 200853 rows with 40 different labels. The news was obtained from 2012-01-28 to 2018-05-26. The data fields are category, headline, authors, link, short description, and date. One drawback of this data is that it is imbalanced on labels. The second dataset is a Chinese news data from toutiao which have 16 unique labels.

Model

News Headline Classification in English News

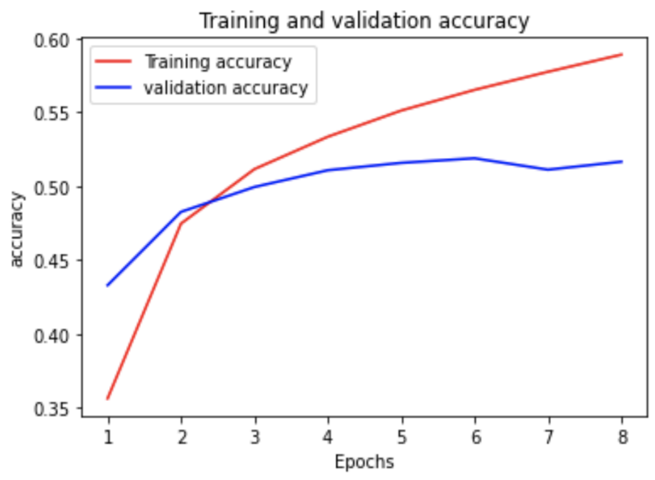
We have created a LSTM model. Long short-term memory (LSTM) is a type of recurrent neural network that is specifically designed to capture long-term dependencies in data. LSTM networks make use of memory cells to store historical information, allowing them to make predictions based on both current and past data. These networks have been shown to perform well on tasks such as language modeling, machine translation, and time series forecasting. In the Keras deep learning framework, LSTM networks can be implemented using the LSTM layer.

The model has an embedding layer that uses a pre-trained embedding matrix with a dimensionality of 50 and is set to be trainable. The embedding layer is followed by three LSTM layers with 64 units each, a dense layer with 64 units and ReLU activation, and a dense output layer with 40 units and softmax activation. The output layer uses L2 regularization with a regularization rate of 0.05. The model is compiled using the Adam optimizer and categorical cross-entropy loss. It is then trained for 8 epochs on the training data and evaluated on the validation data.



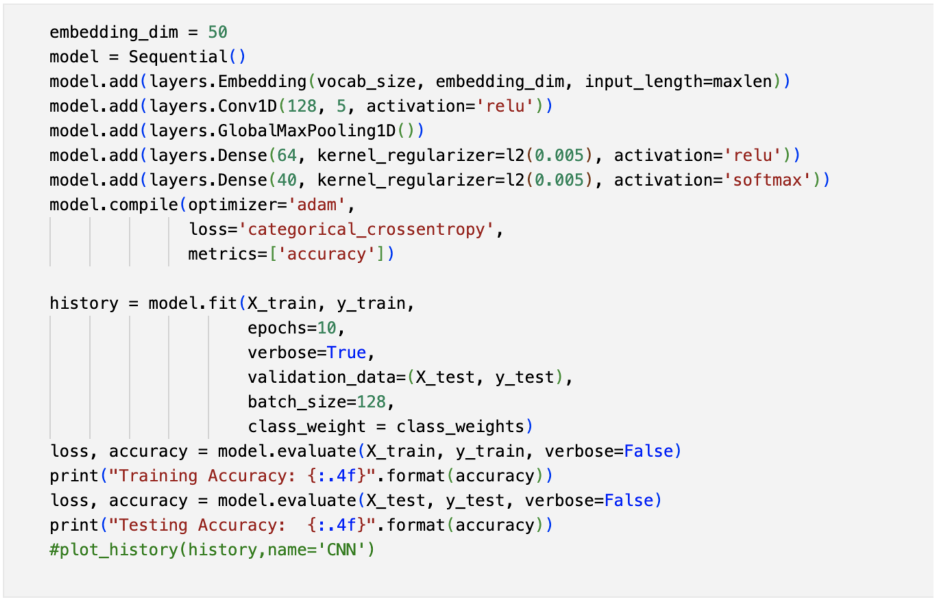
```
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim,
                           weights=[embedding_matrix],
                           input_length=maxlen,
                           #maxlen
                           trainable=True))
model.add(LSTM(64, return_sequences=True))
model.add(LSTM(64, return_sequences=True))
model.add(LSTM(64))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(40, kernel_regularizer=l2(0.05), activation='softmax'))

opt = Adam(lr=0.005)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, y_train,
                    epochs=8,
                    validation_data=(X_test, y_test),
                    batch_size=128,
                    class_weight = class_weights)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

The final loss and accuracy are then calculated on the training data and validation data. The model has a train accuracy of 0.5890 and validation accuracy of 0.5165. The performance of the model is not very good. We assume this is because there are too many labels in the dataset so the dataset is not very balanced.



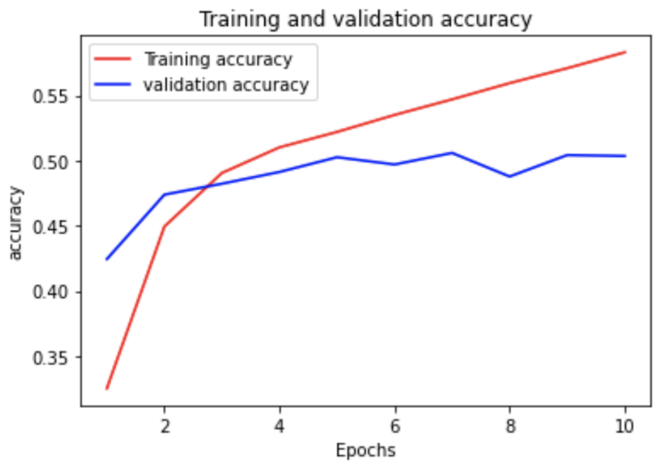
The second model we created is 1D-CNN model. A 1D convolution is a type of convolution operation that can be used to extract local 1D subsequences from sequences. This operation is similar to a 2D convolution, but is applied along the temporal axis of a 1D tensor. By applying the same input transformation to each subsequence of a sequence, 1D convolution layers can recognize local patterns in the sequence and make the resulting model translation invariant. In the Keras deep learning framework, 1D convolution can be implemented using the Conv1D layer, which has a similar interface to the Conv2D layer. A 1D convolutional neural network (1D convnet) is composed of a stack of Conv1D and MaxPooling1D layers, ending with a global pooling layer or a Flatten layer that converts the 3D output of the network into a 2D output for use as input for a classification layer.

The model we created has an embedding layer with a dimensionality of 50, followed by a 1D convolutional layer with 128 filters of size 5, a global max pooling layer, and two dense layers with 64 and 40 units respectively. The final layer uses a softmax activation function to output a probability distribution over the 40 classes. The model is compiled using the Adam optimizer and categorical cross-entropy loss. It is then trained for 10 epochs on the training data and evaluated on the validation data. The final loss and accuracy are then calculated on the training data.

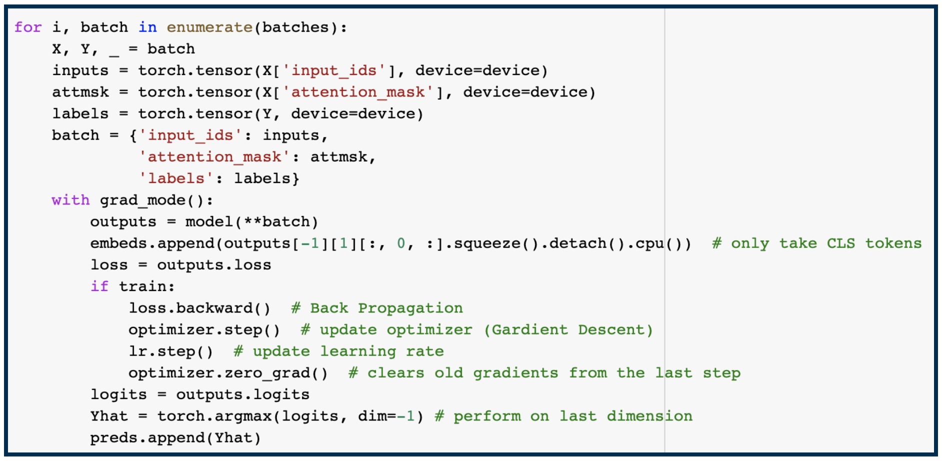


```python
embedding_dim = 50
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim, input_length=maxlen))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(64, kernel_regularizer=l2(0.005), activation='relu'))
model.add(layers.Dense(40, kernel_regularizer=l2(0.005), activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, y_train,
                    epochs=10,
                    verbose=True,
                    validation_data=(X_test, y_test),
                    batch_size=128,
                    class_weight = class_weights)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
#plot_history(history,name='CNN')
```

This model has a training accuracy of 0.5830 and a validation accuracy of 0.5035 which means this ID-CNN model has learned to make predictions on the training data with a high degree of accuracy but has poor generalization ability and is unable to make accurate predictions on new, unseen data. This indicates that the model has likely overfit to the training data and may not be able to make accurate predictions on real-world data. We have used regularization techniques to prevent overfitting, but the result is still not good.



We have also created a Bert model to do the prediction. Before we build and train the model. We have to create batched inputs using Huggingface's BERT tokenizer. So, we create a function batch_data which takes in a dataframe and a batch size, and returns a list of training batches. Each batch is a tuple containing the input data, the ground truth labels, and the headlines. The function iterates over the DataFrame by the specified batch size and creates a batch for each iteration, which includes the text, inputs, and labels for that batch. It then appends the batch to the list of batches and returns the list when all batches have been created. Then we load a pre-trained BERT model. By implementing a training loop from scratch, we can start to train the model, and it has a better performance which accuracy is 0.6666.

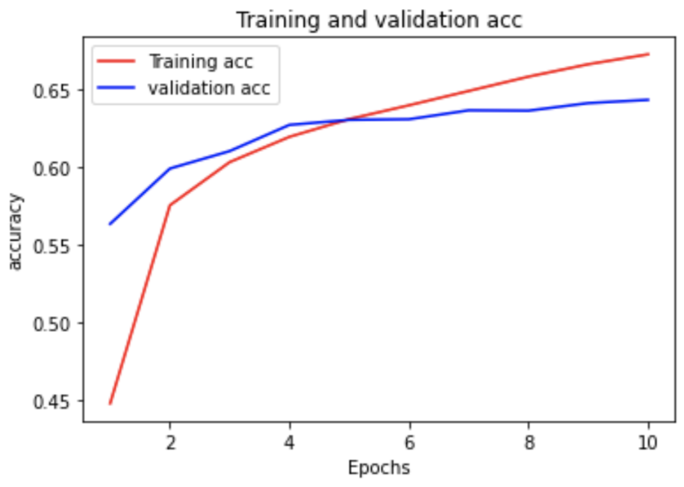


```
for i, batch in enumerate(batches):
    X, Y, _ = batch
    inputs = torch.tensor(X['input_ids'], device=device)
    attmsk = torch.tensor(X['attention_mask'], device=device)
    labels = torch.tensor(Y, device=device)
    batch = {'input_ids': inputs,
             'attention_mask': attmsk,
             'labels': labels}
    with grad_mode():
        outputs = model(**batch)
        embeds.append(outputs[-1][1][:, 0, :].squeeze().detach().cpu())  # only take CLS tokens
        loss = outputs.loss
        if train:
            loss.backward()  # Back Propagation
            optimizer.step()  # update optimizer (Gardient Descent)
            lr.step()  # update learning rate
            optimizer.zero_grad()  # clears old gradients from the last step
        logits = outputs.logits
        Yhat = torch.argmax(logits, dim=-1) # perform on last dimension
        preds.append(Yhat)
```

Hierarchical attention networks (HAN) are a type of neural network that uses stacked recurrent neural networks at the word level, followed by an attention model to identify important words in a sentence and aggregate their representations to form a sentence vector. This vector captures the meaning of the sentence, and the same process is applied to sentence vectors to generate a vector that represents the meaning of an entire document. This vector can then be used for text classification. The attention model in HAN consists of two parts: a bidirectional RNN that learns the meaning behind a sequence of words and returns a vector for each word, and an attention network that uses a shallow neural network to assign weights to each word vector and calculate the weighted sum of these vectors to form a sentence vector. This weighted sum captures the meaning of the sentence. Because HAN uses attention at both the word and sentence level, it is called a hierarchical attention network.

```
l2_reg = l2(0.0001)
word_input = Input(shape=(max_senten_len,), dtype='float32')
word_sequences = embedding_layer(word_input)
word_lstm = Bidirectional(LSTM(150, return_sequences=True, kernel_regularizer=l2_reg))(word_sequences)
word_dense = TimeDistributed(Dense(200, kernel_regularizer=l2_reg))(word_lstm)
word_att = AttentionWithContext()(word_dense)
wordEncoder = Model(word_input, word_att)

sent_input = Input(shape=(max_senten_num, max_senten_len), dtype='float32')
sent_encoder = TimeDistributed(wordEncoder)(sent_input)
sent_lstm = Bidirectional(LSTM(150, return_sequences=True, kernel_regularizer=l2_reg))(sent_encoder)
sent_dense = TimeDistributed(Dense(200, kernel_regularizer=l2_reg))(sent_lstm)
sent_att = Dropout(0.5)(AttentionWithContext()(sent_dense))
preds = Dense(40, activation='softmax')(sent_att)
model = Model(sent_input, preds)
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['acc'])
```

The HAN model has two levels of attention, one at the word level and one at the sentence level. The word-level attention uses a bidirectional LSTM layer to learn the meaning behind a sequence of words, followed by a time-distributed dense layer and an attention layer. The sentence-level attention uses a time-distributed layer to apply the word-level attention to each sentence, followed by a bidirectional LSTM layer, a time-distributed dense layer, and an attention layer with dropout regularization. The final layer is a dense output layer with 40 units and a softmax activation function, and the entire model is compiled using the Adam optimizer and categorical cross-entropy loss. L2 regularization is applied to the kernel weights of some layers in the model with a regularization rate of 0.0000001. The model has a better performance. It has a train accuracy of 0.7580 and validation accuracy of 0.6408.



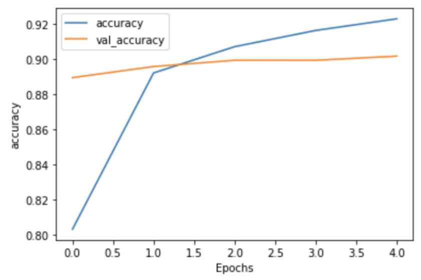
According to the images and results, we can see that most of the models didn’t have a very high accuracy even though we added some regularization to prevent overfitting. To improve the performance of the models. We compress the dataset down to only the 10 labels with the most data without changing the model architecture. The accuracy has a very huge improvement.

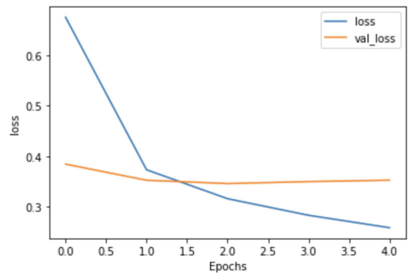
Most of the models can achieve accuracy around 70%, which means the number of the labels or the balance of the dataset has a very huge effect on the model.

News Headline Classification in Chinese News

The headline classification in Chinese news is not always same as the process in English news. The model is the same, which is also LSTM model, but in the text cleaning part, there is a slight different for the word splitting in Chinese. In English we usually use blank space to cut sentences. Full mode gets all the possible words from the sentence. Fast but not accurate.

In the LSTM model, we add a dropout layer. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.





We found that the training accuracy increases as the epoch grows

```
Epoch 1/5
4843/4843 [==============================] - 4502s 928ms/step - loss: 0.6753 - accuracy: 0.8029 - val_loss: 0.3842 - val_accuracy: 0.8894
Epoch 2/5
4843/4843 [==============================] - 4170s 861ms/step - loss: 0.3731 - accuracy: 0.8920 - val_loss: 0.3523 - val_accuracy: 0.8957
Epoch 3/5
4843/4843 [==============================] - 4188s 865ms/step - loss: 0.3158 - accuracy: 0.9070 - val_loss: 0.3458 - val_accuracy: 0.8993
Epoch 4/5
4843/4843 [==============================] - 4140s 855ms/step - loss: 0.2829 - accuracy: 0.9163 - val_loss: 0.3496 - val_accuracy: 0.8993
Epoch 5/5
4843/4843 [==============================] - 4171s 861ms/step - loss: 0.2583 - accuracy: 0.9229 - val_loss: 0.3524 - val_accuracy: 0.9016
```

Here is the model evaluation result.

```
accuracy 0.896247125235208
                    precision    recall  f1-score   support

          news_tech       0.88      0.90      0.89      2759
news_entertainment       0.91      0.92      0.91      3938
        news_sports       0.96      0.95      0.96      3765
           news_car       0.85      0.82      0.83      2716
          news_game       0.93      0.93      0.93      1782
       news_culture       0.95      0.93      0.94      3514
       news_finance       0.90      0.91      0.91      2615
           news_edu       0.88      0.88      0.88      4248
         news_world       0.87      0.89      0.88      2445
      news_military       0.84      0.86      0.85      2191
        news_travel       0.85      0.84      0.84      2706
   news_agriculture       0.88      0.91      0.90      1957
         news_house       0.92      0.93      0.92      2924
         news_story       0.00      0.00      0.00        42
              stock       0.86      0.78      0.82       662

           accuracy                           0.90     38264
          macro avg       0.83      0.83      0.83     38264
       weighted avg       0.90      0.90      0.90     38264
```

Conclusion

We have built a lot of NLP models. For the English new classification, most of the models did a very bad job. In all the models, Bert and HAN have significantly higher test accuracies among all the models. But, after we compress the dataset down to only the 10 labels with the most data without changing the model architecture. All the modes have a huge improvement. For the Chinese model classification, since Chinese is a special language, we have to use a different method to do the tokenization. We use LSTM to do the prediction and the result is very good. We believe this is because the dataset is more balanced. After all the hardworks, we can say we have created some efficient models to do the news headline classification.