Raaghav_94_NLP8_CNN

March 24, 2024

0.1 A Convolutional Neural Network for Modelling Sentences

Implementation of the Paper "A Convolutional Neural Network for Modelling Sentences"

0.2 Importing Libraries

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf

# !pip install datasets
from datasets import load_dataset

from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from sklearn.preprocessing import LabelEncoder

from keras.layers import Layer, InputSpec
from keras.models import Sequential
from keras.layers import Embedding, Dense, Flatten
from keras.layers import Conv1D, GlobalMaxPooling1D

from sklearn.metrics import classification_report

np.random.seed(999)
```

0.3 Load the Data

[ ]:
```
df = load_dataset("sst2")
```

[ ]:
```
X_train = df['train']['sentence'][:-1000]
y_train = np.asarray(df['train']['label'][:-1000])

X_val = df['train']['sentence'][-1000:]
y_val = np.asarray(df['train']['label'][-1000:])

X_test = df['validation']['sentence']
```

```
y_test = np.asarray(df['validation']['label'])
```

0.4 Tokenizing the Text

[ ]:
```
t = Tokenizer(oov_token='<UNK>')
# fit the tokenizer on train documents
t.fit_on_texts(X_train)
t.word_index['<PAD>'] = 0
```

[ ]:
```
X_train = t.texts_to_sequences(X_train)
X_test = t.texts_to_sequences(X_test)
X_val = t.texts_to_sequences(X_val)
```

[ ]:
```
maxlen = 200
num_words = len(t.word_index)

X_train = sequence.pad_sequences(X_train, maxlen=maxlen)
X_test = sequence.pad_sequences(X_test, maxlen=maxlen)
X_val = sequence.pad_sequences(X_val, maxlen=maxlen)
```

0.5 Model Building

[ ]:
```
class Global_k_MaxPooling1D(Layer):
    def __init__(self, k=1, **kwargs):
        super().__init__(**kwargs)
        self.input_spec = InputSpec(ndim=3)
        self.k = k
    def compute_output_shape(self, input_shape):
        return (input_shape[0], (input_shape[1] * self.k))
    def call(self, inputs):
        inputs = tf.transpose(inputs, [0, 2, 1])
        top_k = tf.nn.top_k(inputs, k=self.k, sorted=True, name=None)[0]
        top_k = tf.transpose(top_k, [0, 2, 1])
        return Flatten()(top_k)

def CNN():
    # Conventional CNN for text analysis
    model = Sequential()
    model.add(Embedding(num_words, embedding_dim, input_length=max_len))
    model.add(Conv1D(num_filters, kernel_size, padding='same', strides=1))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(500, activation = 'relu'))
    model.add(Dense(1, activation = 'sigmoid'))
    model.summary()
    return model

def KCNN(k):
```

```
    # CNN using Global_k_max_Pooling1D
    model = Sequential()
    model.add(Embedding(num_words, embedding_dim, input_length=max_len))
    model.add(Conv1D(num_filters, kernel_size, padding='same', strides=1))
    model.add(Global_k_MaxPooling1D(k))
    model.add(Dense(500, activation = 'relu'))
    model.add(Dense(1, activation = 'sigmoid'))
    model.summary()
    return model

def LossPlot():
    fig, loss_ax = plt.subplots()
    acc_ax = loss_ax.twinx()
    loss_ax.plot(history.history['loss'], 'c', linestyle = '-', label='Train␣
 ↪Loss')
    loss_ax.plot(history.history['val_loss'], 'c', linestyle = ':',␣
 ↪label='Validation Loss')
    loss_ax.set_ylim([0.0, 3.0])
    acc_ax.plot(history.history['accuracy'], 'k', linestyle = '-', label='Train␣
 ↪Accuracy')
    acc_ax.plot(history.history['val_accuracy'], 'k', linestyle = ":",␣
 ↪label='Validation Accuracy')
    acc_ax.set_ylim([0.0, 1.0])
    loss_ax.set_xlabel('Epoch', fontsize = 12)
    loss_ax.set_ylabel('Loss', fontsize = 12)
    acc_ax.set_ylabel('Accuracy', fontsize = 12)
    loss_ax.legend(loc='lower left')
    acc_ax.legend(loc='upper left')
    plt.show()
```

[ ]:
```
embedding_dim = 64
num_filters = 128
kernel_size = 3
```

0.6 Model Training

[ ]:
```
model = KCNN(k=3)
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_8 (Embedding)      (None, 200, 64)           884096

 conv1d_8 (Conv1D)            (None, 200, 128)          24704
```

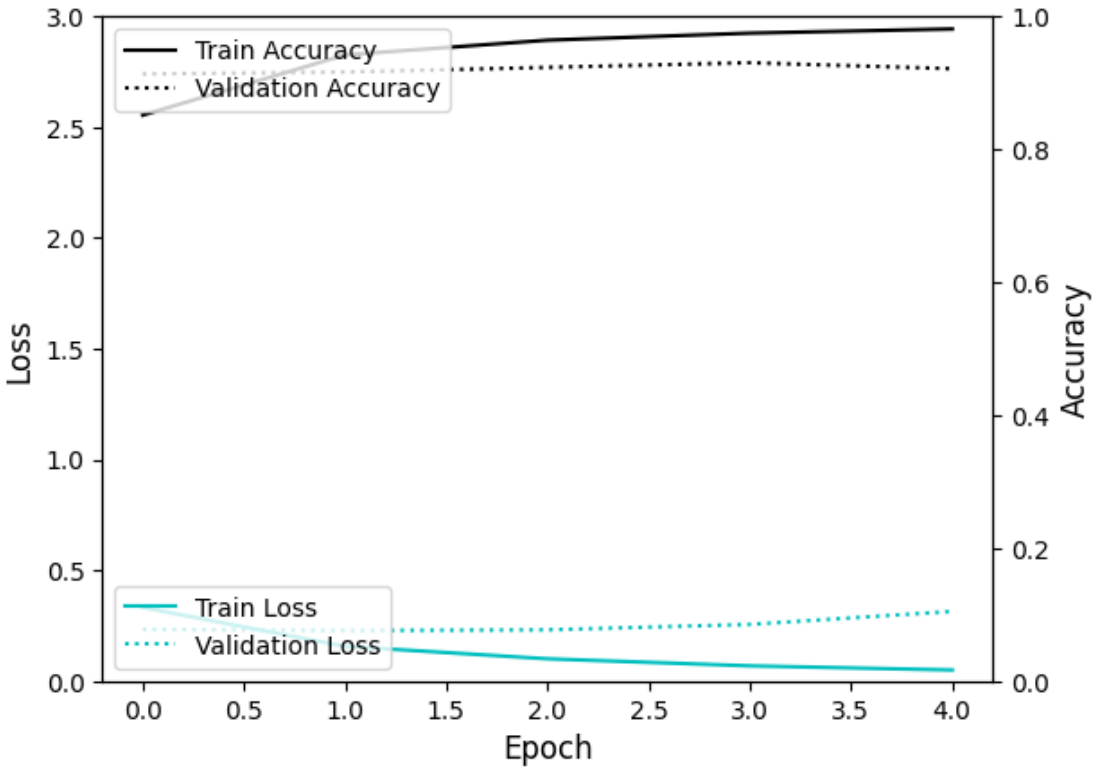
```
 global_k__max_pooling1d_4   (None, 384)              0
 (Global_k_MaxPooling1D)

 dense_14 (Dense)            (None, 500)              192500

 dense_15 (Dense)            (None, 1)                501

=================================================================
Total params: 1101801 (4.20 MB)
Trainable params: 1101801 (4.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
history = model.fit(X_train, y_train, epochs=5, batch_size=128,␣
↪validation_data=(X_val, y_val), verbose=1)
```

```
Epoch 1/5
519/519 [==============================] - 51s 95ms/step - loss: 0.3317 -
accuracy: 0.8511 - val_loss: 0.2330 - val_accuracy: 0.9130
Epoch 2/5
519/519 [==============================] - 22s 42ms/step - loss: 0.1563 -
accuracy: 0.9418 - val_loss: 0.2275 - val_accuracy: 0.9160
Epoch 3/5
519/519 [==============================] - 19s 36ms/step - loss: 0.1000 -
accuracy: 0.9639 - val_loss: 0.2308 - val_accuracy: 0.9230
Epoch 4/5
519/519 [==============================] - 19s 37ms/step - loss: 0.0682 -
accuracy: 0.9745 - val_loss: 0.2552 - val_accuracy: 0.9300
Epoch 5/5
519/519 [==============================] - 20s 39ms/step - loss: 0.0495 -
accuracy: 0.9808 - val_loss: 0.3140 - val_accuracy: 0.9210
```

0.7 Results

[ ]:
```
LossPlot()
```



[ ]:
```
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)

print(classification_report(y_pred, y_test))
```

```
28/28 [==============================] - 0s 12ms/step
              precision    recall  f1-score   support

           0       0.72      0.88      0.79       354
           1       0.90      0.77      0.83       518

    accuracy                           0.81       872
   macro avg       0.81      0.82      0.81       872
weighted avg       0.83      0.81      0.82       872
```

[ ]:
```
test_sentences = t.sequences_to_texts(X_test)
test_sentences = [sent.replace("<UNK>", "").strip() for sent in test_sentences]

for i in range(10):
    print(f"{test_sentences[i]} \nPredicted Label: {y_pred[i][0]} Actual Label:␣
 ↪{y_test[i]}\n")
```

```
it 's a charming and often affecting journey
Predicted Label: 1 Actual Label: 1

bleak and desperate
Predicted Label: 0 Actual Label: 0

allows us to hope that nolan is  to  a major career as a commercial yet
inventive filmmaker
Predicted Label: 1 Actual Label: 1

the acting costumes music cinematography and sound are all astounding given the
production 's  locales
Predicted Label: 1 Actual Label: 1

it 's slow very very slow
Predicted Label: 0 Actual Label: 0

although laced with humor and a few fanciful touches the film is a refreshingly
serious look at young women
Predicted Label: 1 Actual Label: 1

a sometimes tedious film
Predicted Label: 0 Actual Label: 0

or doing last year 's  with your ex wife
Predicted Label: 0 Actual Label: 0

you do n't have to know about music to appreciate the film 's  blend of comedy
and romance
Predicted Label: 1 Actual Label: 1

in exactly 89 minutes most of which passed as slowly as if i 'd been sitting
naked on an  formula 51  from quirky to  to utter turkey
Predicted Label: 0 Actual Label: 0
```