Pytorch' ta CNN Kullanarak Word2Vec ile Eğitilmiş AgNews Verisetine Sınıflandırma Yaparak Kategori Tahmin Etme

```
import pandas as pd

top_data_df = pd.read_csv('agnews/train.csv')
print("Columns in the original dataset:\n")
print(top_data_df.columns)
```

Bu bölümde, verisetimizi okuyoruz ve column larını öğreniyoruz. Bu veri setinde 3 column bulunmaktadır. (Class Index, Title, Description) Class Index 4 farklı label içerir. (1, 2, 3, 4)

OUTPUT:

```
Columns in the original dataset: Index(['Class Index', 'Title', 'Description'],
dtype='object')
```

```
from gensim.utils import simple_preprocess
# Tokenize the text column to get the new column 'tokenized_text'
top_data_df['tokenized_description'] = [simple_preprocess(line, deacc=True) for line
in top_data_df['Description']]
(top_data_df['tokenized_description'].head(10)) #print
```

Her kelimede tek tek dönüşümler yapılabilmesi ve bu kelimelerin sayılara dönüştürülebilmesi için gerekli olan bir adım olan tokenization; cümleleri token(simge) adı verilen sözcük dizilerine böler. (Gensim' in simple_preprocess'i metni küçük harfe dönüştürür ve noktalama işaretlerini kaldırır. Nadir kelimeleri ve en yaygın olarak bu uzunluk aralığına düşen kelimeleri filtrelemeye yardımcı olan minimum ve maksimum uzunluk parametrelerine sahiptir. Dataframe simgelerini almak için kullanılır.)

OUTPUT:

```
0 [reuters, short, sellers, wall, street, dwindl...
1 [reuters, private, investment, firm, carlyle, ...
2 [reuters, soaring, crude, prices, plus, worrie...
3 [reuters, authorities, have, halted, oil, expo...
4 [afp, tearaway, world, oil, prices, toppling, ...
5 [reuters, stocks, ended, slightly, higher, on,...
6 [ap, assets, of, the, nation, retail, money, m...
7 [usatoday, com, retail, sales, bounced, back, ...
8 [forbes, com, after, earning, ph, in, sociolog...
9 [new, york, reuters, short, sellers, wall, str...
Name: tokenized_description, dtype: object
```

```
from gensim.parsing.porter import PorterStemmer
porter_stemmer = PorterStemmer()
# Get the stemmed_tokens
top_data_df['stemmed_tokens'] = [[porter_stemmer.stem(word) for word in tokens] for
                                 tokens in top_data_df['tokenized_description'] ]
(top_data_df['stemmed_tokens'].head(10)) #print
```

Stemming (kök oluşturma), kelimeleri kök kelimesine indirger. Bu işlem sadece ekleri(son/ön) kaldırır. Kök belirleme için basit ve hızlı Porter Stemmer yöntemi kullanılmıştır. Bu kod dataframe üzerinde kök oluşturur ve stemmed_tokens adonda yeni bir cloumn ekler.

OUTPUT:

```
0 [reuter, short, seller, wall, street, dwindl, ...
1 [reuter, privat, invest, firm, carlyl, group, ...
2 [reuter, soar, crude, price, plu, worri, about...
3 [reuter, author, have, halt, oil, export, flow...
4 [afp, tearawai, world, oil, price, toppl, reco...
5 [reuter, stock, end, slightli, higher, on, fri...
6 [ap, asset, of, the, nation, retail, monei, ma...
7 [usatodai, com, retail, sale, bounc, back, bit...
8 [forb, com, after, earn, ph, in, sociolog, dan...
9 [new, york, reuter, short, seller, wall, stree...
Name: stemmed_tokens, dtype: object
```

```
from sklearn.model_selection import train_test_split
# Train Test Split Function
def split_train_test(top_data_df, test_size=0.3, shuffle_state=True):
  X_train, X_test, Y_train, Y_test = train_test_split(top_data_df[['Class Index',
                                     'Title', 'Description', 'stemmed_tokens']],
                              top_data_df['Class Index'],
                              shuffle=shuffle_state,
                              test_size=test_size,
                              random_state=15)
  print("Value counts for Train sentiments")
  print(Y_train.value_counts())
  print("Value counts for Test sentiments")
  print(Y_test.value_counts())
  print(type(X_train))
  print(type(Y_train))
  X_train = X_train.reset_index()
  X_test = X_test.reset_index()
  Y_train = Y_train.to_frame()
  Y_train = Y_train.reset_index()
  Y_test = Y_test.to_frame()
  Y_test = Y_test.reset_index()
  print(X_train.head())
  return X_train, X_test, Y_train, Y_test
# Call the train_test_split
X_train, X_test, Y_train, Y_test = split_train_test(top_data_df)
```

Train verileri, modeli eğitmek için kullanılacak ve test verileri, modelin sınıfları tahmin edeceği verilerdir. Doğruluğu veya diğer model test ölçümlerini kontrol etmek için orijinal etiketlerle karşılaştırılacaktır.

Veriseti; Train verileri %70, Test verileri %30 olacak şekilde ayrılmıştır. Bu işlem için, sckit-learn kütüphanesindeki train_test_split methodu kullanılmıştır. Veriler her sınıf için orantılı olarak dağıtılmıştır ve train ve test teki her label için satır sayısı yazdırılmıştır.

OUTPUT:

```
Value counts for Train sentiments
4 21087
3 21048
2 20940
1 20925
Name: Class Index, dtype: int64
Value counts for Test sentiments
1 9075
2 9060
3 8952
4 8913
Name: Class Index, dtype: int64
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
index ... stemmed_tokens 0 115815 ...
[the, unit, state, on, tuesdai, reject, the, s...
1 13448 ... [reuter, chechen, elect, new, presid, sundai, ...
2 99665 ... [the, death, of, the, vh, video, record, came,...
3 25318 ... [roger, feder, becam, the, first, man, sinc, t...
4 68981 ... [how, do, thei, feel, about, phillip, fulmer, ...
[5 rows x 5 columns]
```

Sinir ağı yalnızca sayısal veriler üzerinde çalıştığı için sözcüklerden sayılara en uygun dönüşüm bulunmalıdır. Girdi olarak 500 boyutlu word2vec vectörleri kullanılmıştır.

yalnızca bir özelliği besliyoruz, yani kelime gömme, böylece conv2d için ilk parametre 1 ve output_channels NUM_FILTERS olacak toplam özellik sayısıdır. Her pencere boyutu için birden fazla filtreye sahip olabiliriz ve bu nedenle bu birçok toplam çıktı olacaktır.

```
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torch
# Use cuda if present
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device available for running: ")
print(device)
```

Ağı oluşturmadan önce eklememiz gereken temel kütüphaneler ve hangi device ı kullandığımız yukarıdaki kod da belirtilmiştir.

```
from gensim.models import Word2Vec
size = 500
window = 3
min_count = 1
workers = 3
sg = 1
OUTPUT_FOLDER = "./OUTPUT_FOLDER/"
# Function to train word2vec model
def make_word2vec_model(top_data_df, padding=True, sg=1, min_count=1, size=500, wo
rkers=3, window=3):
  if  padding:
    print(len(top_data_df))
    temp_df = pd.Series(top_data_df['stemmed_tokens']).values
    temp_df = list(temp_df)
    temp_df.append(['pad'])
    word2vec_file = OUTPUT_FOLDER + 'models/' + 'word2vec_' + str(size) + '_PAD.mo
del'
  else:
    temp_df = top_data_df['stemmed_tokens']
    word2vec_file = OUTPUT_FOLDER + 'models/' + 'word2vec_' + str(size) + '.model'
  w2v_model = Word2Vec(temp_df, min_count = min_count, size = size, workers = work
ers, window = window, sg = sg)

  w2v_model.save(word2vec_file)
  return w2v_model, word2vec_file

# Train Word2vec model
w2vmodel, word2vec_file = make_word2vec_model(top_data_df, padding=True, sg=sg, mi
n_count=min_count, size=size, workers=workers, window=window)
```

Girdi, embedding size ı 500 ile eğitilmiş Word2Vec vektörleridir. Cümlelerin uzunluğunun aynı olması için; cümlenin boyutu, corpus taki en uzun boyuttaki cümleden daha az olduğunda kalan kelimeleri doldurmak için padding token ı kullanılır. Kodda bulunan make_word2vec_model fonksiyonu ile tüm corpus üzerinde Word2Vec modeli eğitilmektedir.

```
max_sen_len = top_data_df.stemmed_tokens.map(len).max()
padding_idx = w2vmodel.wv.vocab['pad'].index
def make_word2vec_vector_cnn(sentence):
  padded_X = [padding_idx for i in range(max_sen_len)]
  i = 0
  for word in sentence:
    if word not in w2vmodel.wv.vocab:
      padded_X[i] = 0
      print(word)
    else:
      padded_X[i] = w2vmodel.wv.vocab[word].index
    i += 1
  return torch.tensor(padded_X, dtype=torch.long, device=device).view(1, -1)
```

Model hazır olduğunda, girdi tensör ü oluşturmak için bir fonksiyon oluşturulmuştur.

```
# Function to get the output tensor
def make_target(label):
  if label == 1:
    return torch.tensor([0], dtype=torch.long, device=device)
  elif label == 2:
    return torch.tensor([1], dtype=torch.long, device=device)
  elif label == 3:
    return torch.tensor([2], dtype=torch.long, device=device)
  else:
    return torch.tensor([3], dtype=torch.long, device=device)
```

Çıkış tensör ü oluşturmak için, etiketten pozitif değerlere eşleme yapılmalıdır. Bu verisetinde labellar pozitiftir ve yapay sinir ağı için uygundur. Çıktı katmanındaki dört nöron, her etiket için olasılıklar verecektir, bu yüzden sadece pozitif sayılarla eşlenmektedir. Bu fonksiyon bunu sağlamaktadır.

```
EMBEDDING_SIZE = 500
NUM_FILTERS = 10
import gensim

class CnnTextClassifier(nn.Module):
  def __init__(self, vocab_size, num_classes, window_sizes=(1,2,3,5)):
    super(CnnTextClassifier, self).__init__()
    w2vmodel = gensim.models.KeyedVectors.load(OUTPUT_FOLDER + 'models/'
                                               + 'word2vec_500_PAD.model')
    weights = w2vmodel.wv
    # With pretrained embeddings
    self.embedding = nn.Embedding.from_pretrained(torch.FloatTensor(weights.vectors),
                                      padding_idx=w2vmodel.wv.vocab['pad'].index)
    # Without pretrained embeddings
    # self.embedding = nn.Embedding(vocab_size, EMBEDDING_SIZE)

    self.convs = nn.ModuleList([
                 nn.Conv2d(1, NUM_FILTERS, [window_size, EMBEDDING_SIZE],
                                                        padding=(window_size - 1, 0))
                 for window_size in window_sizes
    ])

    self.fc = nn.Linear(NUM_FILTERS * len(window_sizes), num_classes)

  def forward(self, x):
    x = self.embedding(x) # [B, T, E]

    # Apply a convolution + max_pool layer for each window size
    x = torch.unsqueeze(x, 1)
    xs = []
    for conv in self.convs:
      x2 = torch.tanh(conv(x))
      x2 = torch.squeeze(x2, -1)
      x2 = F.max_pool1d(x2, x2.size(2))
      xs.append(x2)
    x = torch.cat(xs, 2)

    # FC
    x = x.view(x.size(0), -1)
    logits = self.fc(x)

    probs = F.softmax(logits, dim = 1)

    return probs
```

Bu kod CNN in nasıl tanımlanacağını göstermektedir. Window sizes, number of filters gibi bazı parametreler, farklı sonuçlar elde etmek için değiştirilebilir. Yukarıda oluşturulan model burada yüklenmiştir.

```
NUM_CLASSES = 4
VOCAB_SIZE = len(w2vmodel.wv.vocab)

cnn_model = CnnTextClassifier(vocab_size=VOCAB_SIZE, num_classes=NUM_CLASSES)
cnn_model.to(device)
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(cnn_model.parameters(), lr=0.001)
num_epochs = 30

# Open the file for writing loss
loss_file_name = OUTPUT_FOLDER +  'plots/' + 'cnn_class_big_loss_with_padding.csv'
f = open(loss_file_name,'w')
f.write('iter, loss')
f.write('\n')
losses = []
cnn_model.train()
for epoch in range(num_epochs):
  print("Epoch" + str(epoch + 1))
  train_loss = 0
  for index, row in X_train.iterrows():
    # Clearing the accumulated gradients
    cnn_model.zero_grad()

    # Make the bag of words vector for stemmed tokens
    bow_vec = make_word2vec_vector_cnn(row['stemmed_tokens'])

    # Forward pass to get output
    probs = cnn_model(bow_vec)

    # Get the target label
    target = make_target(Y_train['Class Index'][index])

    # Calculate Loss: softmax --> cross entropy loss
    loss = loss_function(probs, target)
    train_loss += loss.item()

    # Getting gradients w.r.t. parameters
    loss.backward()

    # Updating parameters
    optimizer.step()


  # if index == 0:
  #     continue
  print("Epoch ran :"+ str(epoch+1))
  f.write(str((epoch+1)) + "," + str(train_loss / len(X_train)))
  f.write('\n')
  train_loss = 0

torch.save(cnn_model, OUTPUT_FOLDER + 'cnn_big_model_500_with_padding.pth')

f.close()
print("Input vector")
print(bow_vec.cpu().numpy())
```

Ağ tanımlandıktan sonra, model object, loss ve optimization object gibi training için gerekli nesneler başlatılmaktadır. Bu kod CNN modelinin nasıl eğitileceğini göstermektedir. Epoch sayımız 30 dur. Train verileri için her adımda loss kaydedilmektedir.

```
from sklearn.metrics import classification_report
bow_cnn_predictions = []
original_lables_cnn_bow = []
cnn_model.eval()
loss_df = pd.read_csv(OUTPUT_FOLDER + 'plots/' + 'cnn_class_big_loss_with_padding.
csv')
print(loss_df.columns)
# loss_df.plot('loss')
with torch.no_grad():
  for index, row in X_test.iterrows():
    bow_vec = make_word2vec_vector_cnn(row['stemmed_tokens'])
    probs = cnn_model(bow_vec)
    _, predicted = torch.max(probs.data, 1)
    bow_cnn_predictions.append(predicted.cpu().numpy()[0])
    original_lables_cnn_bow.append(make_target(Y_test['Class Index']
[index]).cpu().numpy()[0])
print(classification_report(original_lables_cnn_bow,bow_cnn_predictions))
loss_file_name = OUTPUT_FOLDER +  'plots/' + 'cnn_class_big_loss_with_padding.csv'
loss_df = pd.read_csv(loss_file_name)
print(loss_df.columns)
plt_500_padding_30_epochs = loss_df[' loss'].plot()
fig = plt_500_padding_30_epochs.get_figure()
#fig.savefig(OUTPUT_FOLDER +'plots/' + 'loss_plt_500_padding_30_epochs.pdf")
```

Bu kodda model kodu test edilmekte ve kayıp grafiği çizdirilmektedir.

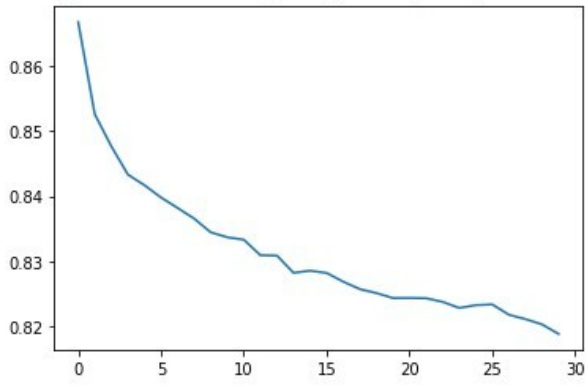
OUTPUT:

```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.93      0.85      0.89      9075
           1       0.94      0.97      0.95      9060
           2       0.86      0.86      0.86      8952
           3       0.84      0.88      0.86      8913

    accuracy                           0.89     36000
   macro avg       0.89      0.89      0.89     36000
weighted avg       0.89      0.89      0.89     36000

Index(['iter', ' loss'], dtype='object')
```



Referanslar

[1] https://towardsdatascience.com/sentiment-classification-using-cnn-in-pytorch-fba3c6840430