Lite

Blog, Short Notes, Course - Code

Natural Language Processing and Computer Vision

Kaggle.com courses.

Person Alpha : PA

Person Beta : PB

PA: Write about NLP?
PB: Using Spacy , three steps: Tokenize that makes token of words, preprocessing for processes including stemming and matching for matching or performing the Natural Language Processing. Natural Language Processing can perform task like Handwriting Recognition.

CODEWISE, Steps involves:

IMPORTING SPACY AND DOCUMENTS,

TOKENIZING

IMPORTING MATCHER

MATCHING

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp("Tea is healthy and calming, don't you think?")
```

```
for token in doc:
    print(token)
```

```
print(f"Token \t\tLemma \t\tStopword".format('Token', 'Lemma', 'Stopword'))
print("-"*40)
for token in doc:
    print(f"{str(token)}\t\t{token.lemma_}\t\t{token.is_stop}")

from spacy.matcher import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab, attr='LOWER')
```

```
text_doc = nlp("Glowing review overall, and some really interesting side-by-si
de "
               "photography tests pitting the iPhone 11 Pro against the "
               "Galaxy Note 10 Plus and last year’s iPhone XS and Google Pixel
3.")
matches = matcher(text_doc)
print(matches)
```

```
match_id, start, end = matches[0]
```

```
print(nlp.vocab.strings[match_id], text_doc[start:end])
```

Persona Alpha:

PA: Text Classification for spacy?
PB: Spam detection and sentimental Analysis are the two techniques involve in text classification.

PA :: List the code you came to know about N.L.P. using text classification model?
PB:

```
import spacy

# Create an empty model
nlp = spacy.blank("en")

# Create the TextCategorizer with exclusive classes and "bow" architecture
textcat = nlp.create_pipe(
              "textcat",
              config={
                "exclusive_classes": True,
                "architecture": "bow"})

# Add the TextCategorizer to the empty model
nlp.add_pipe(textcat)
```

```
# Add labels to text classifier
textcat.add_label("ham")
textcat.add_label("spam")
```

```
train_texts = spam['text'].values
train_labels = [{'cats': {'ham': label == 'ham',
                          'spam': label == 'spam'}}
                for label in spam['label']]
```

```
train_data = list(zip(train_texts, train_labels))
train_data[:3]
```

```
from spacy.util import minibatch

spacy.util.fix_random_seed(1)
optimizer = nlp.begin_training()

# Create the batch generator with batch size = 8
batches = minibatch(train_data, size=8)
```

```
# Iterate through minibatches
for batch in batches:
    # Each batch is a list of (text, label) but we need to
    # send separate lists for texts and labels to update().
    # This is a quick way to split a list of tuples into lists
    texts, labels = zip(*batch)
    nlp.update(texts, labels, sgd=optimizer)
```

```
import random

random.seed(1)
spacy.util.fix_random_seed(1)
optimizer = nlp.begin_training()

losses = {}
for epoch in range(10):
    random.shuffle(train_data)
    # Create the batch generator with batch size = 8
    batches = minibatch(train_data, size=8)
    # Iterate through minibatches
    for batch in batches:
        # Each batch is a list of (text, label) but we need to
        # send separate lists for texts and labels to update().
        # This is a quick way to split a list of tuples into lists
        texts, labels = zip(*batch)
        nlp.update(texts, labels, sgd=optimizer, losses=losses)
    print(losses)
```

```
texts = ["Are you ready for the tea party????? It's gonna be wild",
         "URGENT Reply to this message for GUARANTEED FREE TEA" ]
docs = [nlp.tokenizer(text) for text in texts]

# Use textcat to get the scores for each doc
textcat = nlp.get_pipe('textcat')
scores, _ = textcat.predict(docs)

print(scores)
```

```
# From the scores, find the label with the highest score/probability
predicted_labels = scores.argmax(axis=1)
print([textcat.labels[label] for label in predicted_labels])
```

PA: WORD EMBEDDED MODEL?
PB:

“**Word embeddings** (also called word vectors) represent each word numerically in such a way that the vector corresponds to how that word is used or what it means. Vector encodings are learned by considering the context in which the words appear. Words that appear in similar contexts will have similar vectors. For example, vectors for "leopard", "lion", and "tiger" will be close together, while they'll be far away from "planet" and "castle".”.

PA: Note Down the code for word Embedded Model?
PB:

```
import numpy as np
import spacy

# Need to load the large model to get the vectors
nlp = spacy.load('en_core_web_lg')
```

```
In [2]:
# Disabling other pipes because we don't need them and it'll speed up this part a
bit
text = "These vectors can be used as features for machine learning models."
with nlp.disable_pipes():
    vectors = np.array([token.vector for token in  nlp(text)])
```

```
In [3]:
vectors.shape
```

```
import pandas as pd

# Loading the spam data
# ham is the label for non-spam messages
spam = pd.read_csv('../input/nlp-course/spam.csv')

with nlp.disable_pipes():
    doc_vectors = np.array([nlp(text).vector for text in spam.text])

doc_vectors.shape
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(doc_vectors, spam.label,
                                                    test_size=0.1, random_stat
e=1)
```

```
from sklearn.svm import LinearSVC

# Set dual=False to speed up training, and it's not needed
svc = LinearSVC(random_state=1, dual=False, max_iter=10000)
svc.fit(X_train, y_train)
print(f"Accuracy: {svc.score(X_test, y_test) * 100:.3f}%", )
```

```
def cosine_similarity(a, b):
    return a.dot(b)/np.sqrt(a.dot(a) * b.dot(b))
```

Computer vision.

Convolution , Relu(is equal to ReLU correctly) and CustomConvnets:
Data Augmentation:

Custom Convnets:
Feature extraction through filter , detect and condense.

Example: Input – convolution-relu-pooling-output.

Steps for custom convnet:
Load Data, Define data and train data.

Code of a custom convet:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
```

```
    # First Convolutional Block
    layers.Conv2D(filters=32, kernel_size=5, activation="relu", padding='same'
,
                  # give the input dimensions in the first layer
                  # [height, width, color channels(RGB)]
                  input_shape=[128, 128, 3]),
    layers.MaxPool2D(),

    # Second Convolutional Block
    layers.Conv2D(filters=64, kernel_size=3, activation="relu", padding='same'
),
    layers.MaxPool2D(),

    # Third Convolutional Block
    layers.Conv2D(filters=128, kernel_size=3, activation="relu", padding='same
'),
    layers.MaxPool2D(),

    # Classifier Head
    layers.Flatten(),
    layers.Dense(units=6, activation="relu"),
    layers.Dense(units=1, activation="sigmoid"),
])
model.summary()
model.compile(
    optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
    loss='binary_crossentropy',
    metrics=['binary_accuracy']
)

history = model.fit(
    ds_train,
    validation_data=ds_valid,
    epochs=40,
    verbose=0,
)
```

Data Augmentation:

To increase the data set strength, we might rotate the image, adjust the color or contrast , warping the image and many more things even can be done in combinations.

It is done most of the times online and i.e. while the images are fed in the neural network.Training is done usually in mini batches.

```
from tensorflow import keras
from tensorflow.keras import layers
# these are a new feature in TF 2.2
from tensorflow.keras.layers.experimental import preprocessing


pretrained_base = tf.keras.models.load_model(
    '../input/cv-course-models/cv-course-models/vgg16-pretrained-base',
)
pretrained_base.trainable = False

model = keras.Sequential([
    # Preprocessing
    preprocessing.RandomFlip('horizontal'), # flip left-to-right
    preprocessing.RandomContrast(0.5), # contrast change by up to 50%
    # Base
    pretrained_base,
    # Head
    layers.Flatten(),
    layers.Dense(6, activation='relu'),
    layers.Dense(1, activation='sigmoid'),
])
```

For Relu,write code learned through this course for defining a kernel and apply the kernel and detection of image:

```
import tensorflow as tf

kernel = tf.constant([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1],
])

plt.figure(figsize=(3, 3))
show_kernel(kernel)
image_filter = tf.nn.conv2d(
    input=image,
    filters=kernel,
    # we'll talk about these two in lesson 4!
    strides=1,
    padding='SAME',
)

plt.figure(figsize=(6, 6))
plt.imshow(tf.squeeze(image_filter))
plt.axis('off')
plt.show();
image_detect = tf.nn.relu(image_filter)

plt.figure(figsize=(6, 6))
plt.imshow(tf.squeeze(image_detect))
plt.axis('off')
```

```
plt.show();
```

Detection with Relu:

```
model = keras.Sequential([
    layers.Conv2D(filters=64, kernel_size=3, activation='relu')
    # More layers follow
])
```

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Conv2D(filters=64, kernel_size=3), # activation is None
    # More layers follow
])
```

Kernal are the weights in the C.N.N. during the process.

Since MaxPool2D removes some of the features /pixels , it removes some of the position related information from the map and this property is known as translation invariance. So, the positions of pixel that are provided after the output may not be actual positions of pixel i.e features are not mean to be differentiated from one another by the virtue of position

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Conv2D(filters=64, kernel_size=3), # activation is None
    layers.MaxPool2D(pool_size=2),
    # More layers follow
])
```

Code for Sliding Windows Concept:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Conv2D(filters=64,
                  kernel_size=3,
                  strides=1,
                  padding='same',
                  activation='relu'),
    layers.MaxPool2D(pool_size=2,
                     strides=1,
                     padding='same')
    # More layers follow
])
show_extraction(
    image, kernel,

    # Window parameters
    conv_stride=1,
    pool_size=2,
    pool_stride=2,

    subplot_shape=(1, 4),
```

```
    figsize=(14, 6),
)
show_extraction(
    image, kernel,

    # Window parameters
    conv_stride=3,
    pool_size=2,
    pool_stride=2,

    subplot_shape=(1, 4),
    figsize=(14, 6),
)
```

PA: Application of Computer vision and Knowledge gained through Computer vision course:
PB:
Includes Data augmentation to add more data to the dataset, learned to design own custom convnet with reusable blocks and learned using deep-learning networks to build an image classifier with Keras.