Applied Deep Learning

Project 2
Toxic comment classification

Karolina Szyndler, Michał Filek

1. Dataset

Dataset was downloaded form Kaggle competition titled Toxic Comment Classification Challenge. It consists from 159571 Wikipedia comments. Each element have been labeled by human raters in four categories: toxic, severe toxic, obscene, threat, insult and identity hate.

For solving this problem only toxic category was used. In the whole dataset 16225 comments were labeled as toxic, which is approximately 10% of the data.

2. Preprocessing and embeddings

The text has been preprocessed by converting uppercase letters into lowercase letters, deleting newlines, numbers, special characters and stop words. Then the statements were divided into tokens using keras Tokenizer class.

Dataset was divided into training, validation and test set. Training examples account for 80% of the entire dataset, while the remaining items were divided into test set, containing 500 comments, and validation set covering the remaining examples.

Words have been converted into vector representation using the pre-trained GloVe embeddings. Each word is represented as a vector in 100-dimensional space. 92153 words from the dataset has its representation in GloVe, while 95896 does not. Those words for which no trained vectors were obtained were initialized using the UNK vector. It was calculated as the average of all GloVe vectors.

3. Model architecture and training

Model was trained with:

Batch_size: 30
Epochs: 8
Optimizer: Adam
Loss function: Binary Cross Entropy

Learning scheduler was used, for epochs 1 and 2 learning rate is set to 0.001, for epochs 3, 4 and 5 learning rate is decreased to 0.0001 and for epochs 6, 7 and 8 is decreased to 0.00001.

CNN architecture was used. The first layer creates words embeddings representation. Next, three independent convolutions are applied, each with different filter size: 3, 4 and 5. Convolutions are supposed to find meaningful n-grams.
For each convolution max pooling is applied, outcomes are concatenated and dropout is applied. The result is an input for dense layer with ReLU activation function. In the end batch normalization is applied.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 1000)	0	
embedding_1 (Embedding)	(None, 1000, 100)	18805000	input_2[0][0]
reshape_1 (Reshape)	(None, 1000, 100, 1)	0	embedding_1[0][0]
conv2d_3 (Conv2D)	(None, 998, 1, 512)	154112	reshape_1[0][0]
conv2d_4 (Conv2D)	(None, 997, 1, 512)	205312	reshape_1[0][0]
conv2d_5 (Conv2D)	(None, 996, 1, 512)	256512	reshape_1[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 1, 1, 512)	0	conv2d_3[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 1, 1, 512)	0	conv2d_4[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 512)	0	conv2d_5[0][0]
concatenate_1 (Concatenate)	(None, 3, 1, 512)	0	max_pooling2d_3[0][0] max_pooling2d_4[0][0] max_pooling2d_5[0][0]
flatten_1 (Flatten)	(None, 1536)	0	concatenate_1[0][0]
dropout_1 (Dropout)	(None, 1536)	0	flatten_1[0][0]
dense_2 (Dense)	(None, 246)	378102	dropout_1[0][0]
batch_normalization_v1_1 (Batch	(None, 246)	984	dense_2[0][0]
dense_3 (Dense)	(None, 2)	494	batch_normalization_v1_1[0][0]

Total params: 19,800,516
Trainable params: 19,800,024
Non-trainable params: 492

4. Scores

Train accuracy : 0.9810
Validation accuracy: 0.9596
Test accuracy: 0.9680
Test ROC AUC (we measure it because of unbalanced train samples distribution): 0.9693

5. Deployment on remote host

Trained model was first serialized to *h5* format, then using
tf.keras.experimental.export_saved_model()
to .pb format which is required by tensorflow_serving.
Then default tensorflow_serving docker image was pulled. With model directory within docker image was recreated and pushed on Google Cloud Platform to server image cluster, managed by kubernetes.
We can make model predictions using REST API - server url :
http://35.234.121.157:8501/v1/models/cnn:predict

6. How to run client

I. create virtual env with make_env.sh bash script:

chmod +x make_env.sh; ./make_env.sh; source activate tf_serving

II. run client.py script with proper command line arguments (set test dataset file path, and separator beetween x's and y's, if is different than default sign ';')

python client.py <test.txt path> --separator <'separator_sign'>

III. Accuracy score for passed dataset will be printed