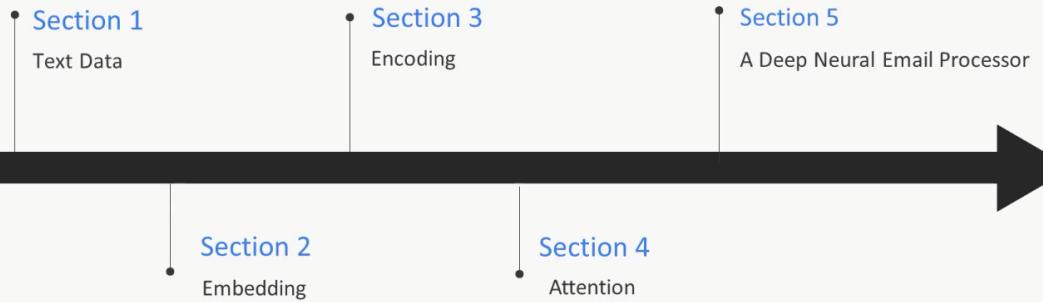


# Tensorflow Solutions for Text

Will Ballard

## The Course Overview

What We'll Learn?



Embedding data is a method of taking text and converting it into numbers, Encoding is where we start using NNs, Attention is an advanced technique for creating more accurate NNs.

```
nbard@horse:~$ tensorboard --logdir=logs/tf_email -p 6006
2017-10-28 13:25:02.707237: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX TITAN X, pci bus id: 0000:03:00.0)
2848/5808 [=====>.....] - ETA: 5s - loss: 0.6604 - acc: 0.5888
```

```
wballard@horse:~
```

			cu_dnnlstm_2[0][0]
dropout_2 (Dropout)	(None, 256)	0	concatenate_1[0][0]
dense_1 (Dense)	(None, 128)	32896	dropout_2[0][0]
dropout_3 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 128)	16512	dropout_3[0][0]
dropout_4 (Dropout)	(None, 128)	0	dense_2[0][0]
dense_3 (Dense)	(None, 2)	258	dropout_4[0][0]
=====	Total params:	59,510,146	
Trainable params:	527,746		
Non-trainable params:	58,982,400		
=====	Train on 5808 samples, validate on 306 samples		
Epoch 1/32			
2017-10-28 13:25:02.707237: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX TITAN X, pci bus id: 0000:03:00.0)			
5808/5808 [=====] - ETA: 0s - loss: 0.5076 - acc: 0.7300 - val_loss: 0.6209 - val_acc: 0.7092			
Epoch 2/32			
5808/5808 [=====] - ETA: 0s - loss: 0.2812 - acc: 0.9060 - val_loss: 0.5243 - val_acc: 0.7157			
Epoch 3/32			
5808/5808 [=====] - ETA: 0s - loss: 0.2515 - acc: 0.9184 - val_loss: 0.5664 - val_acc: 0.6830			
Epoch 4/32			
5808/5808 [=====] - ETA: 0s - loss: 0.2229 - acc: 0.9260 - val_loss: 0.3757 - val_acc: 0.8431			
Epoch 5/32			
5808/5808 [=====] - ETA: 0s - loss: 0.1885 - acc: 0.9372 - val_loss: 0.5027 - val_acc: 0.7451			
Epoch 6/32			
5808/5808 [=====] - ETA: 0s - loss: 0.1727 - acc: 0.9415 - val_loss: 0.4746 - val_acc: 0.8333			
Epoch 7/32			
4288/5808 [=====>.....] - ETA: 1s - loss: 0.1581 - acc: 0.9468			Packt

```
wballard@horse:~
```

			cu_dnnlstm_2[0][0]
Non-trainable params:	58,982,400		
=====	Train on 5808 samples, validate on 306 samples		
Epoch 1/32			
2017-10-28 13:25:02.707237: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX TITAN X, pci bus id: 0000:03:00.0)			
5808/5808 [=====] - ETA: 0s - loss: 0.5076 - acc: 0.7300 - val_loss: 0.6209 - val_acc: 0.7092			
Epoch 2/32			
5808/5808 [=====] - ETA: 0s - loss: 0.2812 - acc: 0.9060 - val_loss: 0.5243 - val_acc: 0.7157			
Epoch 3/32			
5808/5808 [=====] - ETA: 0s - loss: 0.2515 - acc: 0.9184 - val_loss: 0.5664 - val_acc: 0.6830			
Epoch 4/32			
5808/5808 [=====] - ETA: 0s - loss: 0.2229 - acc: 0.9260 - val_loss: 0.3757 - val_acc: 0.8431			
Epoch 5/32			
5808/5808 [=====] - ETA: 0s - loss: 0.1885 - acc: 0.9372 - val_loss: 0.5027 - val_acc: 0.7451			
Epoch 6/32			
5808/5808 [=====] - ETA: 0s - loss: 0.1727 - acc: 0.9415 - val_loss: 0.4746 - val_acc: 0.8333			
Epoch 7/32			
5808/5808 [=====] - ETA: 0s - loss: 0.1526 - acc: 0.9480 - val_loss: 0.4655 - val_acc: 0.8137			
Epoch 8/32			
5808/5808 [=====] - ETA: 0s - loss: 0.1364 - acc: 0.9533 - val_loss: 0.5014 - val_acc: 0.8301			
Epoch 9/32			
5808/5808 [=====] - ETA: 0s - loss: 0.1077 - acc: 0.9611 - val_loss: 0.4429 - val_acc: 0.8235			
Epoch 10/32			
5808/5808 [=====] - ETA: 0s - loss: 0.0960 - acc: 0.9666 - val_loss: 0.7059 - val_acc: 0.7712			
Epoch 11/32			
5808/5808 [=====] - ETA: 0s - loss: 0.0858 - acc: 0.9683 - val_loss: 0.7048 - val_acc: 0.7353			
Epoch 12/32			
5808/5808 [=====] - ETA: 0s - loss: 0.0656 - acc: 0.9793 - val_loss: 1.0534 - val_acc: 0.7222			
Epoch 13/32			
5808/5808 [=====] - ETA: 0s - loss: 0.0767 - acc: 0.9721 - val_loss: 0.6633 - val_acc: 0.7124			
Epoch 14/32			
5808/5808 [=====] - ETA: 0s - loss: 0.0482 - acc: 0.9845 - val_loss: 0.7150 - val_acc: 0.8268			
Epoch 15/32			
2240/5808 [=====>.....] - ETA: 2s - loss: 0.0241 - acc: 0.9938			Packt

```
replies_ensemble.py examples
4 # one hot encode the labels
5 targets = replies.one_hot_labels
6
7 # now set up sequencing and embedding
8 sources = replies.texts
9
10 HIDDEN = 32
11 ACTIVATION = 'selu'
12 INITIALIZER = 'lecun_normal'
13
14 # data is loaded and preprocessed, now create a keras model to encode
15 import keras
16
17 inputs = keras.layers.TrigramInput(xlen))
18 # embedding to turn self.trigram = CharacterTrigramEmbedding()
19 embedded = replies.trigram.model(inputs)
20
21
22 # plain old dense
23 dense = keras.layers.Dense(HIDDEN, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(embedded)
24 dense = keras.layers.Dropout(0.5)(dense)
25 dense = keras.layers.Dense(HIDDEN, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(dense)
26 dense = keras.layers.Dropout(0.5)(dense)
27
28 # convolution to learn word and phrase like features
29 conv = keras.layers.Conv1D(HIDDEN, 3, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(embedded)
30 conv = keras.layers.MaxPooling1D(3)(conv)
31 conv = keras.layers.Conv1D(HIDDEN, 3, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(conv)
32 conv = keras.layers.MaxPooling1D(3)(conv)
33
34 # recurrent with attention, this generates sequences
35 # note this is GPU only, and keras
36 # '>=2.0.9, it is shocking slow otherwise
37 recurrent_forward = keras.layers.CuDNNLSTM(HIDDEN, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(conv)
38 recurrent_backward = keras.layers.CuDNNLSTM(HIDDEN, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(mailscanner.layers.Pickle)
39 recurrent = keras.layers.concatenate([recurrent_forward, recurrent_backward])
40
41
```

```
replies_ensemble.py examples
20 embedded = replies.trigram.model(inputs)
21
22
23 # plain old dense
24 dense = keras.layers.Dense(HIDDEN, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(embedded)
25 dense = keras.layers.Dropout(0.5)(dense)
26 dense = keras.layers.Dense(HIDDEN, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(dense)
27 dense = keras.layers.Dropout(0.5)(dense)
28
29 # convolution to learn word and phrase like features
30 conv = keras.layers.Conv1D(HIDDEN, 3, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(embedded)
31 conv = keras.layers.MaxPooling1D(3)(conv)
32 conv = keras.layers.Conv1D(HIDDEN, 3, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(conv)
33 conv = keras.layers.MaxPooling1D(3)(conv)
34
35 # recurrent with attention, this generates sequences
36 # note this is GPU only, and keras
37 # '>=2.0.9, it is shocking slow otherwise
38 recurrent_forward = keras.layers.CuDNNLSTM(HIDDEN, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(conv)
39 recurrent_backward = keras.layers.CuDNNLSTM(HIDDEN, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(mailscanner.layers.Pickle)
40 recurrent = keras.layers.concatenate([recurrent_forward, recurrent_backward])
41
42 # now attend to the most important
43 self_attention = mailscanner.layers.TimeDistributedSelfAttention(activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)
44 time_attention = mailscanner.layers.SelfAttention(activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)
45
46 # now make a consistent shape and ensemble together as a stack, using global max pooling
47 # to take out any remaining time steps and keep the strongest signals
48 dense = keras.layers.GlobalMaxPooling1D()(dense)
49 conv = keras.layers.GlobalMaxPooling1D()(conv)
50 self_attention = keras.layers.GlobalMaxPooling1D()(self_attention)
51 time_attention = keras.layers.GlobalMaxPooling1D()(time_attention)
52 ensemble = keras.layers.concatenate([dense, conv, recurrent, self_attention, time_attention])
53
54
```

```
replies_ensemble.py examples
51 time_attention = keras.layers.GlobalMaxPooling1D()(time_attention)
52 ensemble = keras.layers.concatenate([dense, conv, recurrent, self_attention, time_attention])
53
54
55 # dense before final output
56 stack = keras.layers.Dense(HIDDEN, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(ensemble)
57 stack = keras.layers.Dropout(0.5)(stack)
58 stack = keras.layers.Dense(HIDDEN, activation=ACTIVATION, kernel_regularizer=keras.regularizers.l2(0.), kernel_initializer=INITIALIZER)(stack)
59 stack = keras.layers.Dropout(0.5)(stack)
60
61 # softmax on two classes -- which map to our 0, 1 onehots
62 outputs = keras.layers.Dense(2, activation='softmax')(stack)
63
64 model = keras.models.Model(inputs=inputs, outputs=outputs)
65 model.compile(
66     loss='categorical_crossentropy',
67     optimizer='adam',
68     metrics=['accuracy']
69 )
70 model.summary()
71
72 # model was impressive in overfitting -- hitting 100% accuracy in train
73 # so -- change the mix shift of the data, more for testing, less data for overfitting
74 # moving even less data to validation, meaning more training data --> less overfitting
75 model.fit(
76     x=sources,
77     y=targets,
78     validation_split=0.01,
79     batch_size=128,
80     epochs=256
81 )
82
```

## Prerequisites

- Experience with command line shell
- Python scripting or application development

## Course Goals

- Build machine learning environments
- Understand how to prepare data for machine learning
- Create classical, convolutional and deep neural networks
- Serve neural network models via REST

# Embedding

## Section 2

In this Section, we are going to take a look at...

- Representing words with embeddings
- Pretrained word embeddings with Keras
- Pretrained character embeddings with Keras
- Pretrained FastText embeddings with Keras

# Representing Words with Embeddings

In this Video, we are going to take a look at...

- Why embedding
- Dimensionality reduction
- Embedding algorithms
- How embedding learns

**Dimensionality Reduction** is related to enhancing computability as well as improving your model's ability to learn.

# Why Embed?

Generalization and Computability

Pure text is a very high dimension space when one-hot, and can easily overfit

Very large dimension tensors for sequences of words easily exceed memory

Bag-of-words loses word order!

## Dimensionality Reduction

400,000 Vocabulary

100 Word Document

150MB

300 Dimension Embedding

100 Word Document

100KB

We want to be able to create models that understand that words exist in a sequence and not just count words like the classic bag of words model

## Embedding Algorithms

- GLOVE: <https://nlp.stanford.edu/projects/glove/>
- Word2Vec: <https://arxiv.org/abs/1301.3781>
- FastText: <https://github.com/facebookresearch/fastText>

FastText actually builds relationships out of the character positions inside of words. We generally have a set of about 100 - 300 positions that are the embeddings generated out of the numbers.

## Learning Embedding

- All about context, learn the surrounding words
- Given words, predict the next word
- Give words with one removed, fill in the blank
- Given a pair of words, predict the probability of them being near

The embedding algorithms generally learn the context of words by building a sort of language model of the surrounding words. There are three main embedding strategies as listed above that provide an unsupervised model approach to train a model that understands language and words.

<https://github.com/wballard/vectoria/tree/embedding>

# Encoding

## Section 3

We will now move from looking at words to looking at entire emails.

In this Section, we are going to take a look at...

- Assembling email and embeddings
- Dense neural networks on text
- Convolutional neural networks on text
- Recurrent neural networks on text

We will take an entire email collection we downloaded earlier and turn that into a dataset for machine learning. We then build 3 different ML networks that attempt to predict whether or not the recipient will reply to an email mainly based on its text.

## Assembling Email with Embeddings

In this Video, we are going to take a look at...

- Detecting which email generated a reply
- Generating a replied/did-no-reply dataset
- LabeledTextFileDataset

We will look at all the sent emails, look at their identifiers and then find the emails to which they replied. We then generate this list as a text file dataset to which we store the label and all the text in the particular email. We then create a class **LabeledTextFileDataset** that will load these emails into memory in an embedding format.

[https://github.com/wballard/mailscanner/tree/section\\_3](https://github.com/wballard/mailscanner/tree/section_3)

```

#!/usr/bin/env python
...
3  prepare-replies-dataset
4
5 Usage:
6     | prepare-replies-dataset <email_database> <dataset_text>
7
8 Prepare a text dataset from email replies, each line will be:
9     0 <tab> text of email without reply
10    1 <tab> text of email with reply
11 ...
12
13 import re
14
15 import docopt
16
17 import mailscanner
18
19 if __name__ == '__main__':
20     arguments = docopt.docopt(__doc__, version=mailscanner.__version__)
21     gdb = mailscanner.EmailDatabase(arguments['<email_database>'])
22     replies = mailscanner.datasets.RepliedToDataset(gdb)
23     scrub = re.compile('[\t\r\n]')
24     with open(arguments['<dataset_text>'], 'w') as dataset_text:
25         for (reply, text) in replies.dataset:
26             text = scrub.sub(' ', text)
27             dataset_text.write('{0}\t{1}\n'.format(reply, text))

```

Let us see code to generate a text file from our emails using the tool **docopt**

```

# databases.py mailscanner
1 ...
2 Adapters to store an individual user's email in a database.
3 ...
4
5 import os
6 import sqlite3
7
8 from tqdm import tqdm
9
10
11 class EmailDatabase(sqlite3.Connection):
12     ...
13     Store email in raw RFC822 format with identifiers.
14     ...
15
16     def __init__(self, database_filename):
17         ...
18         Parameters
19         -----
20         database_filename
21             A string, where to store the file on disk. The folder must exist.
22             ...
23         if not os.path.exists(database_filename):
24             creation_query = """
25                 create table all_email(id text, body text);
26                 create unique index all_email_id on all_email(id);
27                 create table sent_email(id text, body text);
28                 create unique index sent_email_id on sent_email(id);
29             """

```

```

# databases.py mailscanner
26         create unique index all_email_id on all_email(id);
27         create table sent_email(id text, body text);
28         create unique index sent_email_id on sent_email(id);
29         ...
30     else:
31         creation_query = None
32     super(EmailDatabase, self).__init__(database_filename)
33     if creation_query:
34         self.executescript(creation_query)
35
36
37     def sent(self, visitor, verbose=True):
38         ...
39         Visit all sent emails.
40
41         Parameters
42         -----
43         visitor
44             A callable that receives each email text.
45             ...
46         cursor = self.cursor()
47         cursor.execute('select count(*) from sent_email')
48         count = cursor.fetchall()[0][0]
49         cursor.execute('select body from sent_email')
50         for row in tqdm(cursor.fetchall(), total=count, desc="Sent", unit='email', disable=(not verbose)):
51             visitor(row[0])
52
53     def all(self, visitor, verbose=True):
54         ...
55         Visit all emails.
56
57         Parameters
58         -----
59         visitor
60             A callable that receives each email text.
61             ...

```

```
database.py mailsScanner
1 Visit all sent emails.
2
3 Parameters
4 -----
5 visitor
6     A callable that receives each email text.
7 ...
8 cursor = self.cursor()
9 cursor.execute('select count(*) from sent_email')
10 count = cursor.fetchall()[0][0]
11 cursor.execute('select body from sent_email')
12 for row in tqdm(cursor.fetchall(), total=count, desc="Sent", unit='email', disable=(not verbose)):
13     visitor(row[0])
14
15 def all(self, visitor, verbose=True):
16     ...
17     Visit all emails.
18
19     Parameters
20     -----
21     visitor
22         A callable that receives each email text.
23 ...
24 cursor = self.cursor()
25 cursor.execute('select count(*) from all_email')
26 count = cursor.fetchall()[0][0]
27 cursor.execute('select body from all_email')
28 for row in tqdm(cursor, total=count, desc="All", unit='email', disable=(not verbose)):
29     visitor(row[0])
30
```

```
replies.py mailsScanner/datasets
1 ...
2 Create a dataset to learn which emails you are likely to reply to.
3 ...
4
5 from ..parser import parse
6
7
8 class RepliedToDataset:
9     ...
10    Build a dataset of emails that generated replies, along with a balanced number of
11    negative samples that did not generate a reply.
12
13 This visits every sent email in the provided database, extracts identifiers of emails that
14 generated replies, and hashes them.
15
16 With a hash of all emails that generated replies in hand, all received emails are visited
17 in order to extract the text of the email for each reply generating message. The very next
18 non-reply-generating message encountered is used as a negative sample to offset and generate
19 balanced classes.
20
21 Attributes
22 -----
23 dataset
24     A list of (Replied|DidNotReply, email text) tuples.
25 ...
26
27 def __init__(self, email_database):
28     ...
29     Parameters
30     -----
31     email_database
32         Visit this database to create training samples.
33     ...
34     replied_to = {}
35
36     def is_a_reply(email):
37
38         reply = parse(email).get('In-Reply-To', None)
39         if reply:
40             replied_to[reply] = True
41
42         email_database.sent(is_a_reply)
43
44         self.dataset = []
45
46         def extract_replies(email):
47             email = parse(email)
48             if replied_to.get(email.get('Message-ID'), False):
49                 # a message that generated a reply!
50                 self.dataset.append(('Replied', ' '.join(map(str, email.values()))))
51             return
52
53             # if we get here, this was not a reply, use it as a negative sample
54             # if we have an odd number of entries to balance out
55             if len(self.dataset) % 2 == 1:
56                 self.dataset.append(('DidNotReply', ' '.join(map(str, email.values()))))
57
58         email_database.all(extract_replies)
```

Packt

```
replies.py mailsScanner/datasets
1
2     Parameters
3     -----
4     email_database
5         Visit this database to create training samples.
6     ...
7     replied_to = {}
8
9     def is_a_reply(email):
10
11         reply = parse(email).get('In-Reply-To', None)
12         if reply:
13             replied_to[reply] = True
14
15         email_database.sent(is_a_reply)
16
17         self.dataset = []
18
19         def extract_replies(email):
20             email = parse(email)
21             if replied_to.get(email.get('Message-ID'), False):
22                 # a message that generated a reply!
23                 self.dataset.append(('Replied', ' '.join(map(str, email.values()))))
24             return
25
26             # if we get here, this was not a reply, use it as a negative sample
27             # if we have an odd number of entries to balance out
28             if len(self.dataset) % 2 == 1:
29                 self.dataset.append(('DidNotReply', ' '.join(map(str, email.values()))))
30
31         email_database.all(extract_replies)
```

```
wballard (e) root .../git/wballard/mailscanner > # section_3
$ ls
bin examples gmail.db mailscanner mailscanner.egg-info Makefile README.md replies.txt setup.py var
wballard (e) root .../git/wballard/mailscanner > # section_3
$ ./bin/prepare-replies-dataset ./gmail.db
Usage:
  prepare-replies-dataset <email_database> <dataset_text>
wballard (e) root .../git/wballard/mailscanner > # section_3
1 $ ./bin/prepare-replies-dataset ./gmail.db replies.txt
Sent: 14%|██████████| 1019/7448 [00:01<00:09, 645.64email/s]
wballard (e) root .../git/wballard/mailscanner > # section_3

  prepare-replies-dataset <email_database> <dataset_text>
wballard (e) root .../git/wballard/mailscanner > # section_3
1 $ ./bin/prepare-replies-dataset ./gmail.db replies.txt
Sent: 100%|██████████| 7448/7448 [00:24<00:00, 298.02email/s]
All: 39%|██████████| 91627/234594 [04:32<05:33, 428.56email/s]
wballard (e) root .../git/wballard/mailscanner > # section_3
$ 

[cc. 100%]
wballard (e) root .../git/wballard/mailscanner > # section_3
$ head -n 1 ./replies.txt
Replied ('', 'wballard@mailframe.net') by 10.143.31.6 with HTTP; Sun, 15 Jun 2008 18:34:34 -0700 (PDT) <busey@duels.co
m; neutral (google.com: 72.14.204.169 is neither permitted nor denied by best guess record for domain of busey@duels.c
om) client-ip=72.14.204.169; mx.google.com; spf=neutral (google.com: 72.14.204.169 is neither permitted nor denied by
best guess record for domain of busey@duels.com) smtp.mail=busey@duels.com <42f35d82086151834x70245a08g78e3122c117bf9
fd@mail.gmail.com> (2008, 6, 15, 20, 34, 34, 0, 1, -1, -18000) ('Andrew Busey', 'busey@duels.com') ('Will Ballard', 'w
ballard@mailframe.net') Draft 1.0 multipart/mixed; boundary="-----_Part_15607_6752670.1213580074705" --_Duels - A
Player vs. Player Strategy Game Challenge me: www.duels.com/loki Corporate info: www.challengegames.com <br cle
ar="all"><br>--<br>Duels - A Player vs. Player Strategy Game<br>Challenge me:&nbsp;&nbsp;<a href="http://www.duels.co
m/loki">www.duels.com/loki</a><br>Corporate info:&nbsp;&nbsp;<a href="http://www.challengegames.com">www.challengegame
s.com</a>
wballard (e) root .../git/wballard/mailscanner > # section_3
$ 
```

We can then check the first line in the replies.txt file

The screenshot shows a code editor interface with two open files:

- labeled.txt** (top editor):
 

```
1 Good  Good stuff!
2 Bad   Bad stuff!
```
- textfiles.py** (bottom editor):
 

```
17 -----
18     labels
19         A binary encoded array of the labels
20     one_hot_labels
21         One hot encoding of labels
22     texts
23         A 2-d tensor of string and ngram positional sequence encodings.
24
25     At a given index `n` `labels[n]` is the corresponding label for `texts[n]`.
26
27     >>> import mailscanner
28     >>> dataset = mailscanner.datasets.LabeledTextFileDataset('./var/data/labeled.txt')
29     >>> dataset.labels
30     array([1, 0])
31     >>> dataset.texts
32     array([[112284,  8220, 64853, ...,  0,    0,    0],
33            [107523,  82916, 185037, ...,  0,    0,    0]], dtype=int32)
34
35     def __init__(self, textfile_path):
36         ...
37             Read the text, and separate it.
38
39             Parameters
40             -----
41             textfile_path
42                 A string path, which is passed to `smart_open`, so this can be a local file
43                 or even an S3 url.
44             ...
45             label_buffer = []
46             text_buffer = []
47             for line in smart_open(textfile_path):
48                 label, text = line.decode('utf8').split('\t')
49                 label_buffer.append(label)
50                 text_buffer.append(text.encode())
```

```

1  ...
2  Turn text files on disk into in memory tensors for use with machine learning.
3  ...
4
5  from sklearn.pipeline import Pipeline
6  from sklearn.preprocessing import LabelBinarizer, LabelEncoder, OneHotEncoder
7  from smart_open import smart_open
8  from vectoria import CharacterTrigramEmbedding
9
10
11 class LabeledTextFileDataset:
12     ...
13     Read text files with one string per line, of the form:
14     <label> <tab> <text>
15
16     Attributes
17     -----
18     labels
19         A binary encoded array of the labels
20     one_hot_labels
21         One hot encoding of labels
22     texts
23         A 2-d tensor of string and ngram positional sequence encodings.
24
25     At a given index `n` `labels[n]` is the corresponding label for `texts[n]`.
26
27     >>> import mailscanner
28     >>> dataset = mailscanner.datasets.LabeledTextFileDataset('./var/data/labeled.txt')
29     >>> dataset.labels
30     array([1, 0])
31     >>> dataset.texts
32     array([[1112284, 8220, 64853, ..., 0, 0, 0],
33            [107523, 82916, 185037, ..., 0, 0, 0]], dtype=int32)
34
35     def __init__(self, textfile_path):
36

```

Packt

```

29
30     >>> dataset.labels
31     array([1, 0])
32     >>> dataset.texts
33     array([[1112284, 8220, 64853, ..., 0, 0, 0],
34            [107523, 82916, 185037, ..., 0, 0, 0]], dtype=int32)
35
36     def __init__(self, textfile_path):
37
38         Read the text, and separate it.
39
40         Parameters
41         -----
42         textfile_path
43             A string path, which is passed to `smart_open`, so this can be a local file
44             or even an S3 url.
45
46         label_buffer = []
47         text_buffer = []
48         for line in smart_open(textfile_path):
49             label, text = line.decode('utf8').split('\t')
50             label_buffer.append(label)
51             text_buffer.append(text.strip())
52
53         label_encoder = LabelEncoder()
54         onehot_encoder = Pipeline([
55             ('binarizer', LabelBinarizer()),
56             ('onehot', OneHotEncoder())
57         ])
58         self.labels = label_encoder.fit_transform(label_buffer)
59         # mildly tricky, need to wrap the array in an array
60         self.one_hot_labels = onehot_encoder.fit_transform(self.labels).toarray()
61         self.trigram = CharacterTrigramEmbedding()
62         self.texts = self.trigram.sequencer.transform(text_buffer)
63

```

# Attention

## Section 4

What is attention in a neural network? Demonstrate attention, Learn about the features of a full featured Keras layer and Code Self-Attention.

In this Section, we are going to take a look at...

- SelfAttention
- Time series self attention
- Ensemble networks on text

# Self Attention

In this Video, we are going to take a look at...

- What is attention
- Full featured Keras layer implementations
- Self Attention implementation

<https://github.com/wballard/mailscanner/tree/attention>

## Attention!

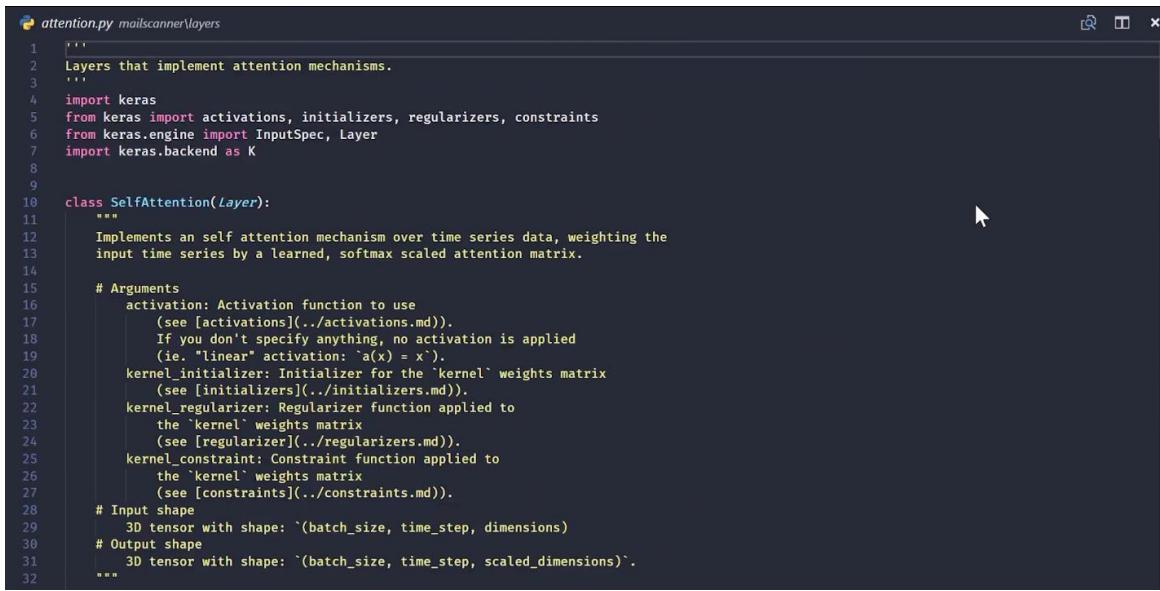
Focus on the important

Softmax used at the interior of the model

Self normalizing as softmax sum approaches one

Effectively the probability each time step is important, bringing ‘attention’ to only those steps

Attention is a way to get your model to focus on the most important features.



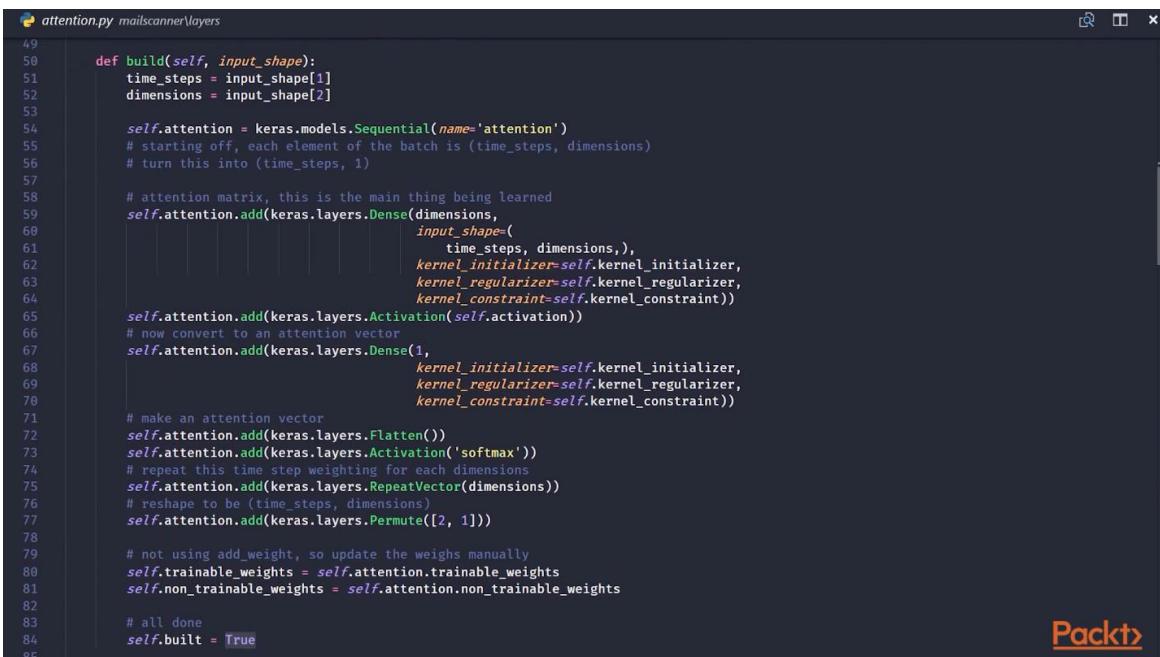
```
attention.py mailscanner\layers
1 """
2     Layers that implement attention mechanisms.
3 """
4
5 import keras
6 from keras import activations, initializers, regularizers, constraints
7 from keras.engine import InputSpec, Layer
8 import keras.backend as K
9
10
11 class SelfAttention(Layer):
12     """
13         Implements an self attention mechanism over time series data, weighting the
14         input time series by a learned, softmax scaled attention matrix.
15
16         # Arguments
17             activation: Activation function to use
18                 (see [activations](./activations.md)).
19                 If you don't specify anything, no activation is applied
20                 (ie. "linear" activation:  $a(x) = x$ ).
21             kernel_initializer: Initializer for the 'kernel' weights matrix
22                 (see [initializers](./initializers.md)).
23             kernel_regularizer: Regularizer function applied to
24                 the 'kernel' weights matrix
25                 (see [regularizer](./regularizers.md)).
26             kernel_constraint: Constraint function applied to
27                 the 'kernel' weights matrix
28                 (see [constraints](./constraints.md)).
29
30             # Input shape
31             3D tensor with shape: `(batch_size, time_step, dimensions)`
32             # Output shape
33             3D tensor with shape: `(batch_size, time_step, scaled_dimensions)`.
```

The **keras.backend** gives us direct access to TF's backend.

```

32 """
33
34     def __init__(self,
35         activation=None,
36         kernel_initializer='glorot_uniform',
37         kernel_regularizer=None,
38         kernel_constraint=None,
39         **kwargs):
40         if 'input_shape' not in kwargs and 'input_dim' in kwargs:
41             kwargs['input_shape'] = (kwargs.pop('input_dim'),)
42         super(SelfAttention, self).__init__(**kwargs)
43         self.activation = activations.get(activation)
44         self.kernel_initializer = initializers.get(kernel_initializer)
45         self.kernel_regularizer = regularizers.get(kernel_regularizer)
46         self.kernel_constraint = constraints.get(kernel_constraint)
47         self.input_spec = InputSpec(ndim=3)
48         self.supports_masking = True
49
50     def build(self, input_shape):
51         time_steps = input_shape[1]

```

attention.py mailscanner/layers

```

49
50     def build(self, input_shape):
51         time_steps = input_shape[1]
52         dimensions = input_shape[2]
53
54         self.attention = keras.models.Sequential(name='attention')
55         # starting off, each element of the batch is (time_steps, dimensions)
56         # turn this into (time_steps, 1)
57
58         # attention matrix, this is the main thing being learned
59         self.attention.add(keras.layers.Dense(dimensions,
60                                             input_shape=(
61                                                 time_steps, dimensions),
62                                             kernel_initializer=self.kernel_initializer,
63                                             kernel_regularizer=self.kernel_regularizer,
64                                             kernel_constraint=self.kernel_constraint))
65         self.attention.add(keras.layers.Activation(self.activation))
66         # now convert to an attention vector
67         self.attention.add(keras.layers.Dense(1,
68                                             kernel_initializer=self.kernel_initializer,
69                                             kernel_regularizer=self.kernel_regularizer,
70                                             kernel_constraint=self.kernel_constraint))
71         # make an attention vector
72         self.attention.add(keras.layers.Flatten())
73         self.attention.add(keras.layers.Activation('softmax'))
74         # repeat this time step weighting for each dimensions
75         self.attention.add(keras.layers.RepeatVector(dimensions))
76         # reshape to be (time_steps, dimensions)
77         self.attention.add(keras.layers.Permute([2, 1]))
78
79         # not using add_weight, so update the weights manually
80         self.trainable_weights = self.attention.trainable_weights
81         self.non_trainable_weights = self.attention.non_trainable_weights
82
83         # all done
84         self.built = True
85

```

Packt

```

86     self.built = True
87
88     def call(self, inputs):
89         # build the attention matrix
90         attention = self.attention(inputs)
91         # apply the attention matrix with element wise multiplication
92         return keras.layers.Multiply()([inputs, attention])
93
94     def compute_output_shape(self, input_shape):
95         # there is no change in shape, the values are just weighted
96         return input_shape
97
98     def get_config(self):
99         config = {
100             'activation': activations.serialize(self.activation),
101             'kernel_initializer': initializers.serialize(self.kernel_initializer),
102             'kernel_regularizer': regularizers.serialize(self.kernel_regularizer),
103             'kernel_constraint': constraints.serialize(self.kernel_constraint),
104         }
105         return dict(config)

```

Packt

# A Deep Neural Email Processor

## Section 5

In this Section, we are going to take a look at...

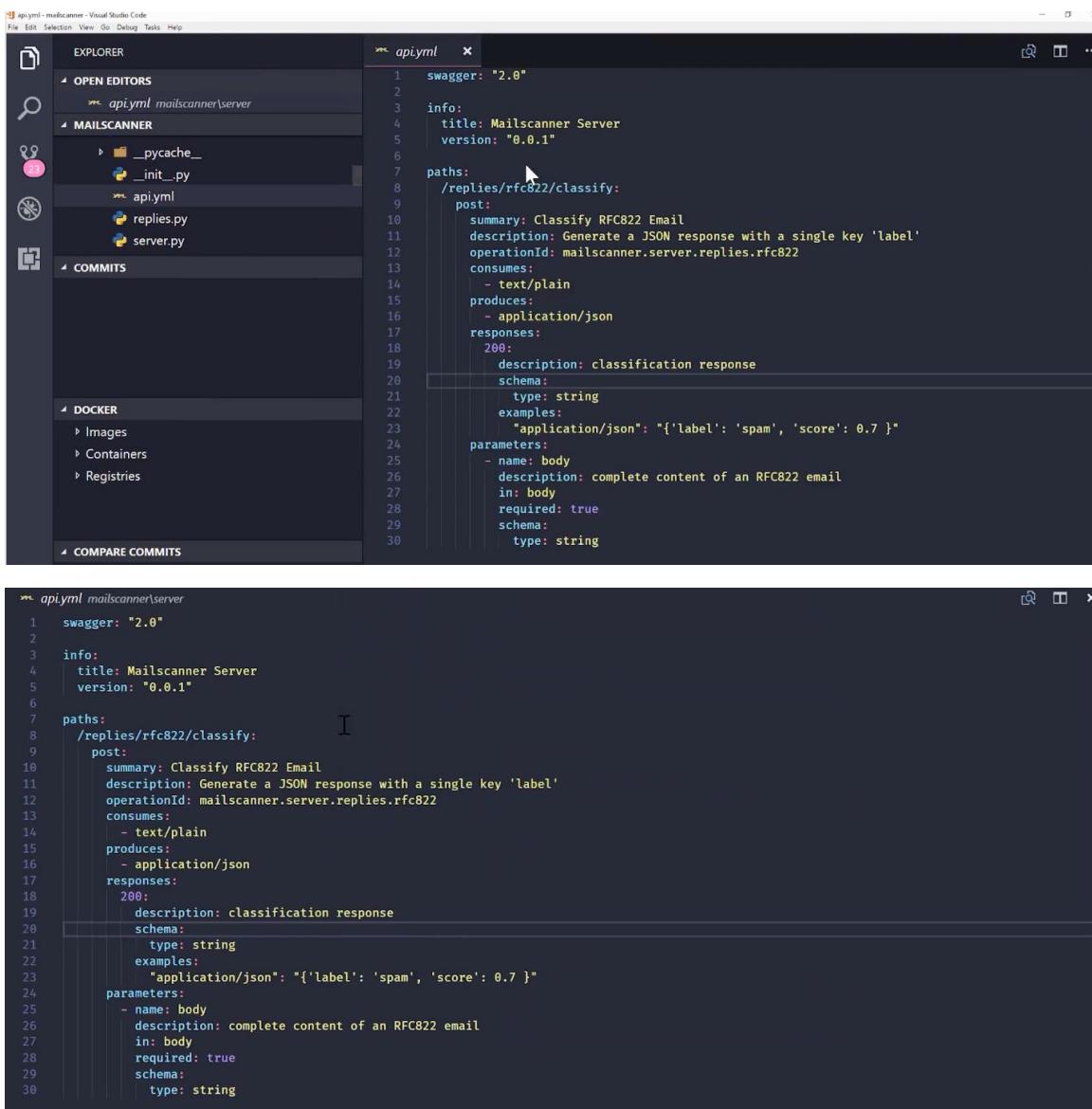
- REST API definition
- Trained models in Docker containers
- Prediction server in Docker containers

# REST API Definition

In this Video, we are going to take a look at...

- Getting project source code
- OpenAPI/Swagger with Connexion
- Connexion server
- Connexion handler and model

<https://github.com/wballard/mailscanner/tree/server>



The screenshot shows the Visual Studio Code interface with two open files:

- api.yaml - mailscanner - Visual Studio Code**: This is the main file being edited. It contains the OpenAPI specification for the MailScanner Server. The code is as follows:

```
swagger: "2.0"
info:
  title: Mailscanner Server
  version: "0.0.1"
paths:
  /replies/rfc822/classify:
    post:
      summary: Classify RFC822 Email
      description: Generate a JSON response with a single key 'label'
      operationId: mailscanner.server.replies.rfc822
      consumes:
        - text/plain
      produces:
        - application/json
      responses:
        200:
          description: classification response
          schema:
            type: string
          examples:
            "application/json": "{ 'label': 'spam', 'score': 0.7 }"
      parameters:
        - name: body
          description: complete content of an RFC822 email
          in: body
          required: true
          schema:
            type: string
```

- api.yaml mailscanner\server**: This is a copy of the same file located in the mailscanner\server folder. It shows the identical content.

```
server.py mailsScanner\server
1  ...
2 Server module, this can be used from the command line or via uWSGI
3  ...
4
5 import os
6
7 import connexion
8 import docopt
9
10 import mailsScanner
11
12 PORT = os.environ.get('PORT', 5000)
13 DEBUG = os.environ.get('DEBUG', False)
14
15 SERVER_IN = os.path.dirname(os.path.abspath(__file__))
16 # WSGI module level variable
17 application = connexion.App(__name__, port=PORT, specification_dir=SERVER_IN)
18 application.add_api('api.yml')
19 mailsScanner.server.replies.load_model_codec(
20     os.path.join(SERVER_IN, '../..var/data/replies.weight'),
21     os.path.join(SERVER_IN, '../..var/data/replies.pickle')
22 )
23
24 if __name__ == '__main__':
25     application.run(debug=DEBUG)
26
```

```
replies.py mailsScanner\server
1  ...
2 Individual email handling methods.
3  ...
4
5 from ..datasets import LabeledTextFileDataset
6 from ..models import Ensemble
7
8 # preload this, it has a large tensor inside
9 # connexion only allows module level functions as handlers
10 # so module level caching of large data is required
11 # -- beats loading in on every request!
12 MODEL = None
13 CODEC = None
14
15 def load_model_codec(path_to_weights, path_to_codec):
16     ...
17     Load up the module level variables for the codec/dataset
18     and the trained machine learning model.
19     ...
20     global CODEC, MODEL
21     print('loading codec from', path_to_codec)
22     CODEC = LabeledTextFileDataset.load(path_to_codec)
23     print('loading weights from', path_to_weights)
24     MODEL = Ensemble(CODEC)
25     MODEL.load_weights(path_to_weights)
26     print(MODEL.summary())
27
28
29 def rfc822(body):
30     ...
31     Parameters
32     -----
33     body
34         An entire RFC822 string.
35
36     Returns
37     -----
```

Packt

```
replies.py mailsScanner\server
20
21     global CODEC, MODEL
22     print('loading codec from', path_to_codec)
23     CODEC = LabeledTextFileDataset.load(path_to_codec)
24     print('loading weights from', path_to_weights)
25     MODEL = Ensemble(CODEC)
26     MODEL.load_weights(path_to_weights)
27     print(MODEL.summary())
28
29 def rfc822(body):
30     ...
31     Parameters
32     -----
33     body
34         An entire RFC822 string.
35
36     Returns
37     -----
38     string
39         JSON string encoding the classification result.
40
41     # text, sequenced as ngram, ready to be predicted
42     body = body.decode('utf8')
43     sequenced = CODEC.trigram.sequencer.transform([body])
44     predicted = MODEL.predict(sequenced)
45     decode = CODEC.decode_prediction(predicted[0])
46
47     return {
48         'label': decode[0],
49         # cast off the numpy type
50         'score': float(decode[1])
51     }
```

Packt