SUPERBUG
AI BUG MANAGEMENT SYSTEM

Rohit Taneja & Gaurav Shad

Table of Contents

Introduction to Superbug 3
Problem 3
Idea 3
High Level Design 4
Dataset 4
Solution Stack 4
Getting started with Tool usage 6
Alexa Integration 8
Software Design 10
Control Flow 10
Source code 11
Model architecture and accuracy results 13
Environment Setup and testing 15
Future Work 16
Improvement in model accuracy 16
Improving Integration with Alexa 16
Appendix 16
Website 16
Docker container for testing the software 16
Dataset source 16
Word embeddings 16

Introduction to Superbug

Problem

One problem that plagues productivity organizations irrespective of their size is how to know what bugs exist in upcoming products or what are the top bug reports or feature requests that needs to get implemented or fixed to have a product remain competitive in the market. A successful product entails continuous attention to backend as well as front-end with minimal latency in bug fix. Neglect of important features or bug fixes often results in subpar customer experience which in-turn leads to the organization losing its competitive advantage in the market. Thus, an agile redressal of bugs distinguishes a successful company and product from a not-so successful one. To provide and sustain a successful product, a nimble and efficient bug-tracking system should constitute the heart of the product life-cycle. In every organization, Engineers, Product Managers, Sales Personnel - all strive to track bugs for their respective needs and customers but often are unable to keep up with challenging deadlines and multiple projects at hand. Another recurrent scenario in organizations is; one team detects a bug and works on it for months to provide a solution, however when a different customer reports a similar issue, it gets reported to a different person or team. This results in duplication of efforts, high turnaround time and productivity loss. It's a very arduous task especially in large organizations, for humans or existing tracking mechanisms to have a complete view of all bugs in the system, where duplicate bugs could be identified and connectivity between teams be ensured to enable them to collaborate and share insights and experiences working on similar bugs. Also, some of the high priority bugs or feature requests can easily fall into cracks because current tracking systems involve human intervention to track bugs and requires engineers/QA to manually enter status updates.

Idea

- Design an Artificial Intelligent System based on deep learning technology to identify duplicate bug reports.
- Use the pre-trained model to classify new bugs and automatically assign them to the right team/module.
- Voice bug reporting system using Alexa
- Voice bug tracking and reminder system using Alexa – Ask Alexa to give a status about bug report or enable Alexa to remind/update team if no activity is seen on that bug report for a defined period.

High Level Design

Dataset

We will be using a bug triaging dataset from Mozilla; https://bugzilla.mozilla.org to train a model that will work as an algorithm for this project. We are training with 4 classes or labels, one per each module in the Mozilla Firefox application. We have kept the data balance for training by choosing almost same number of samples for each class, so that bias in the training is removed due to dataset.

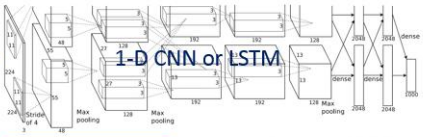
Label	Module	Number of samples
0	General	4822
1	Preferences	5160
2	Theme	4623
3	Toolbars and Customization	5517

We have extracted the synopsis and description for each module and stored in our training server in different folders; one folder per label. Each sample length is quite small and thus pose a challenge in training, that is why we have explored training with 2 different models; 1 dimensional Convolutional Network as well as Recurrent Neural Network LSTM. We believe that testing with longer length sample size helps in higher accuracy in training.

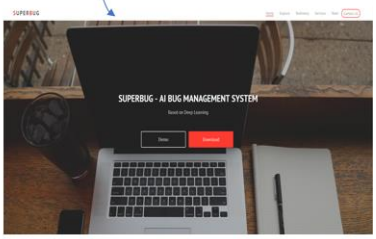
Solution Stack

Component	Type	Version/Comments
Deep Learning Framework	Keras with Tensorflow Backend	TF 1.2.1
Hardware	Nvidia GPU	GeForce GTX
UI	www.super-bug.com	Website provides interface to predict class/label for new bug report
Voice control	Alexa Integration	

Bug data from
Bugzilla for Mozilla
project



Inference on our
website

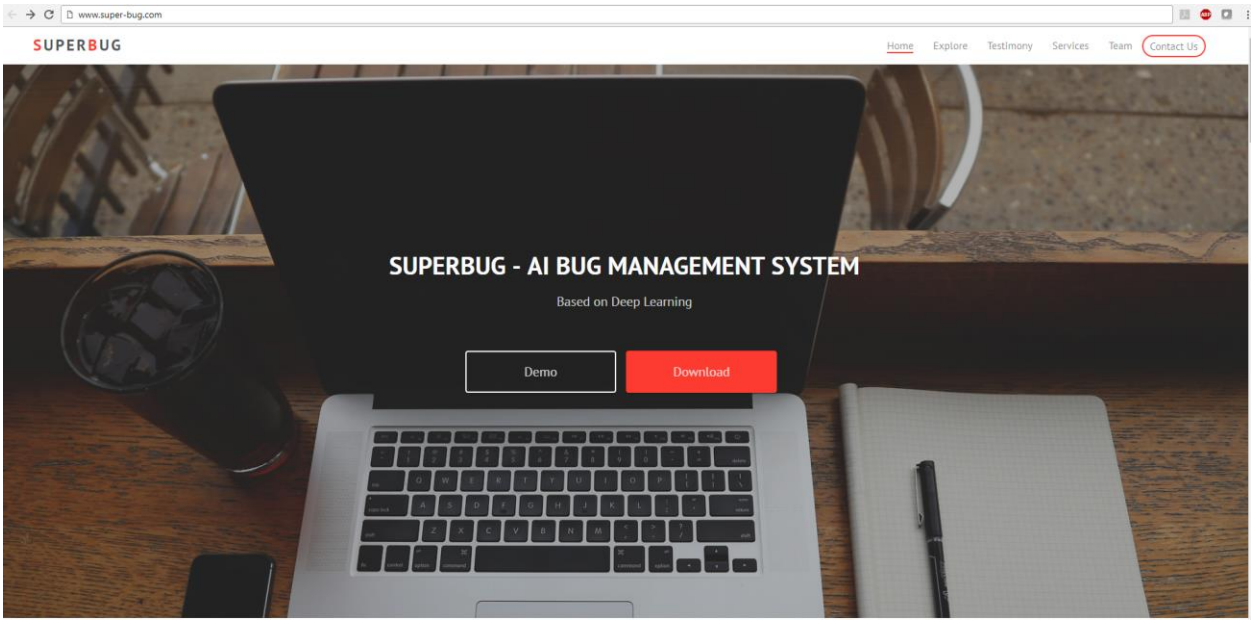


www.super-bug.com

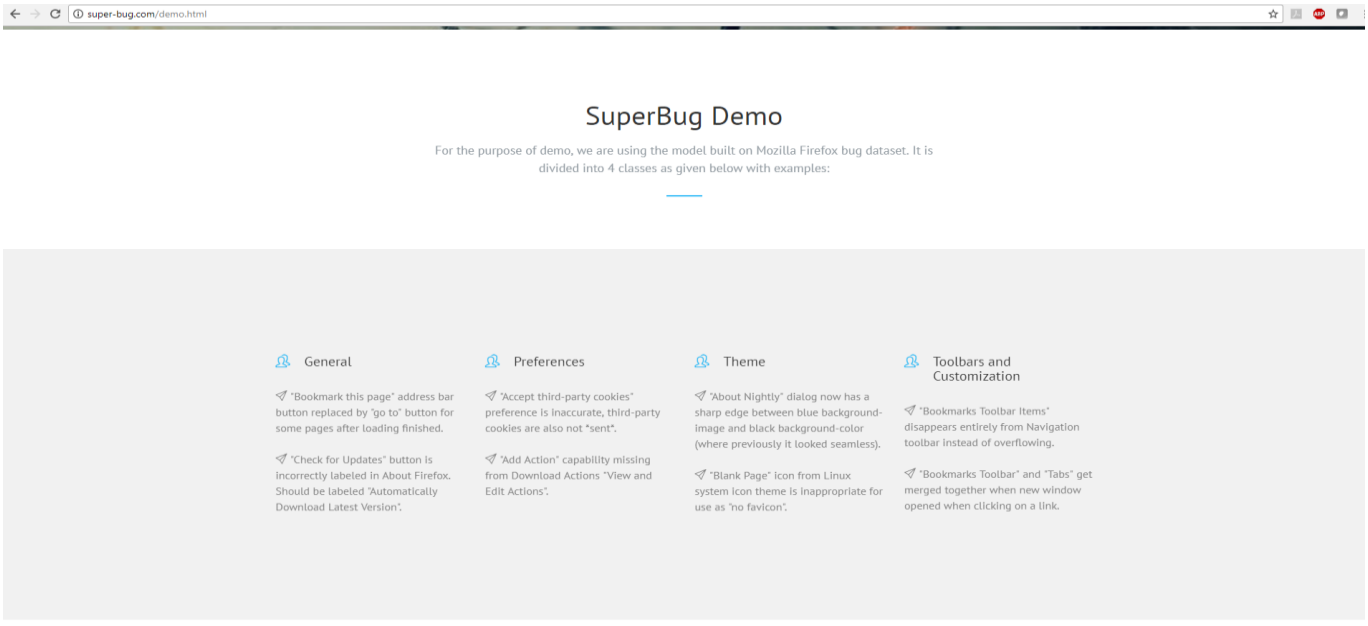
Getting started with Tool usage

www.super-bug.com hosts a inference engine which connects at back-end to our trained model on mozilla bugzilla data set. Here are the steps to navigate to the demo page:

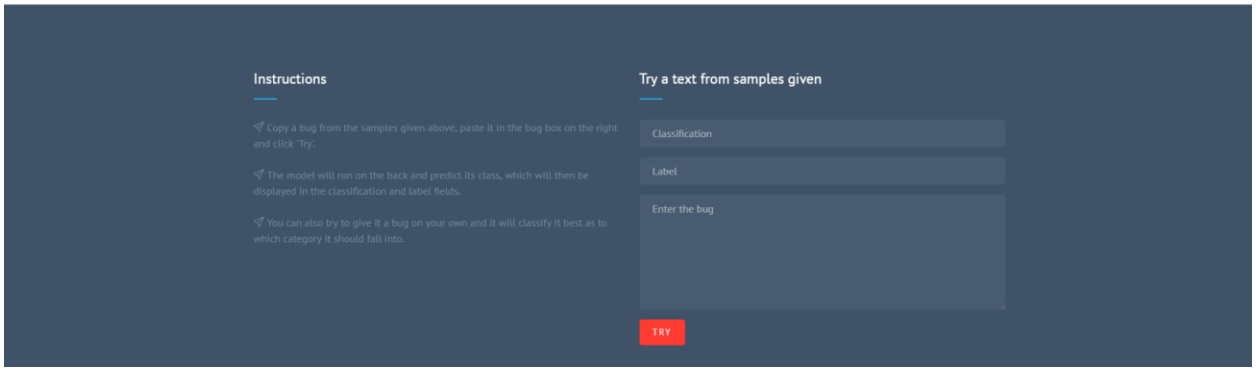
1. Starting page



2. Clicking on Demo will take you to:



3. At the bottom of Demo page, you can paste the new bug report in “Enter the bug” space and when you click try, it will perform inference with our trained model and predict the class/module it belongs to.



Alexa Integration

To run this skill, you need to do two things. The first is to deploy the example code in lambda, and the second is to configure the Alexa skill to use Lambda.

AWS Lambda Setup

1. Go to the AWS Console and click on the Lambda link. Note: ensure you are in us-east or you wont be able to use Alexa with Lambda.
2. Click on the Create a Lambda Function or Get Started Now button.
3. Skip the blueprint
4. Name the Lambda Function "Interview".
5. Select the runtime as Java 8
6. Go to the the /alexa-skills-EHR/ directory containing pom.xml, and run 'mvn assembly:assembly -DdescriptorId=jar-with-dependencies package'. This will generate a zip file named "alexa-skills-kit-samples-1.0-jar-with-dependencies.jar" in the target directory.
7. Select Code entry type as "Upload a .ZIP file" and then upload the "alexa-skills-kit-samples-1.0-jar-with-dependencies.jar" file from the build directory to Lambda
8. Set the Handler as EHRSystem.EHRsystemSpeechletRequestStreamHandler (this refers to the Lambda RequestStreamHandler file in the zip).
9. Create a basic execution role and click create.
10. Leave the Advanced settings as the defaults.
11. Click "Next" and review the settings then click "Create Function"
12. Click the "Event Sources" tab and select "Add event source"
13. Set the Event Source type as Alexa Skills kit and Enable it now. Click Submit.
14. Copy the ARN from the top right to be used later in the Alexa Skill Setup.

Alexa Skill Setup

1. Go to the Alexa Console and click Add a New Skill.
2. Set "Interview" as the skill name and "Interview" as the invocation name, this is what is used to activate the skill. For example you would say: "Alexa, start superbug".
3. Select the Lambda ARN for the skill Endpoint and paste the ARN copied from above. Click Next.

4. Copy the Intent Schema from the included IntentSchema.json.
5. Copy the Sample Utterances from the included SampleUtterances.txt. Click Next.
6. Go back to the skill Information tab and copy the appId. Paste the appId into the EHRSystemSpeechletRequestStreamHandler.java file for the variable supportedApplicationIds, then update the lambda source zip file with this change and upload to lambda again, this step makes sure the lambda function only serves request from authorized source.
7. You are now able to start using the skill! You should be able to go to the Echo webpage and see the skill enabled.
8. In order to test it, try to say some of the Sample Utterances from the Examples section below.
9. The skill is now saved and once you are finished testing you can continue to use it.

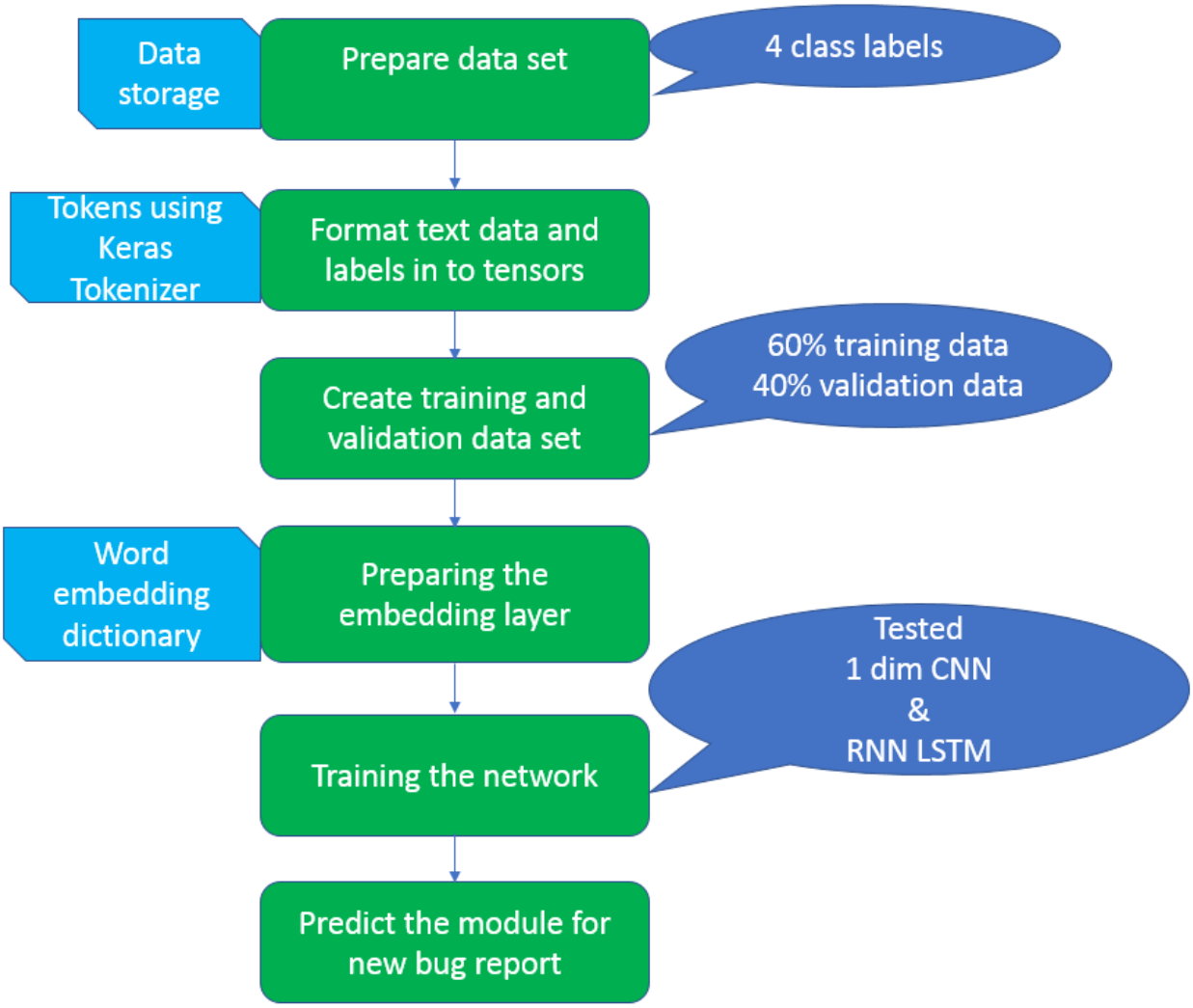
Examples

One-shot model:

```
User: "Alexa, start superbug."
Alexa: "Welcome to superbug!"
```

Software Design

Control Flow



Source code

Only the model architecture snippet is shown here and rest of the code is attached as the jupyter notebook:

```
#### Training the network ####
# We have created 2 models - 1) 1-Dim CNN (Conv1D) 2) RNN LSTM
# Conv1-D is commented out as we observe better accuracy with LSTM for current hyper-parameter
# and embedding tunings. There exists some scope of improevement for both models.

from keras.layers import Activation, Dense, Input, Conv1D, MaxPooling1D, Flatten, Dropout
from keras.callbacks import EarlyStopping
from keras.layers.wrappers import Bidirectional
from keras.models import Model
from keras.models import Sequential
from keras.layers import LSTM
from keras.optimizers import RMSprop



## Model 1 - Using 1D Convolutional Neural Network ##
## This layer creates a convolution kernel   ##
## that is convolved with the layer input    ##
## over a single spatial (or temporal)       ##
## dimension to produce a tensor of outputs. ##
## 3 conv1D (ReLu activation) + Maxpooling + Dropout layers    ##


#sequence_input = Input(shape=(100,), dtype='int32')
#embedded_sequences = embedding_layer(sequence_input)
#print embedded_sequences
#x = Conv1D(128,5, activation='tanh')(embedded_sequences)
#x = MaxPooling1D(5)(x)
#x = Dropout(0.75)(x)
#x = Conv1D(128,5, activation='tanh')(x)
#x = MaxPooling1D(5)(x)
#x = Dropout(0.75)(x)
#x = Conv1D(128,2, activation='tanh')(x)
#x = MaxPooling1D(2)(x)
#x = Dropout(0.75)(x)
#x = Flatten()(x)
```

```
#x = Dense(128, activation='tanh')(x)
#preds = Dense(len(labels_index), activation='softmax')(x)
#model = Model(sequence_input, preds)
# return_sequences = true


## Model 1 - Using RNN LSTM (Bidirectional) ##
## This layer creates a convolution kernel   ##
## that is convolved with the layer input    ##
## over a single spatial (or temporal)       ##
## dimension to produce a tensor of outputs. ##
## 2 LStM(ReLu activation) + Maxpooling + Dropout layers    ##

model = Sequential()
model.add(Embedding(len(word_index) + 1, 100, weights=[embedding_matrix],
input_length=46, trainable=True))
model.add(Bidirectional(LSTM(128, return_sequences=True, activation='relu')))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(128, activation='relu')))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(len(labels_index),activation='softmax'))
model.add(Activation('softmax'))


#Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])

print model.summary() # Print model architechture with parameters

#es = EarlyStopping(patience=5)  # this was used during training for monitoring
overfitting
                        # (gap between training vs validataionaccuracy)


#Begin training: 40 epochs and batch size of 128
model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=100, batch_size=128)
scores = model.evaluate(x_val, y_val) # train and Val scores
print('Test score:', scores[0])
print('Test accuracy:', scores[1])

print("Accuracy: %.2f%%" % (scores[1]*100))
```

Model architecture and accuracy results

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 46, 100)	1101900
lstm_5 (LSTM)	(None, 46, 128)	117248
dropout_5 (Dropout)	(None, 46, 128)	0
lstm_6 (LSTM)	(None, 128)	131584
dropout_6 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 128)	16512
dense_6 (Dense)	(None, 4)	516
activation_3 (Activation)	(None, 4)	0

Total params: 1,367,760
Trainable params: 1,367,760
Non-trainable params: 0

We achieve~ 75% validation accuracy within 40 epochs. We have tried with different epoch settings like trying with 100 epochs but convergence doesn't change around after 40 epochs.

```
Train on 16098 samples, validate on 4024 samples
Epoch 1/100
16098/16098 [==============================] - 30s - loss: 1.2727 - acc: 0.4355 - val_loss: 1.1626 - val_acc: 0.5731
Epoch 2/100
16098/16098 [==============================] - 30s - loss: 1.1206 - acc: 0.6221 - val_loss: 1.1767 - val_acc: 0.5624
Epoch 3/100
16098/16098 [==============================] - 30s - loss: 1.2877 - acc: 0.4519 - val_loss: 1.1937 - val_acc: 0.5413
Epoch 4/100
16098/16098 [==============================] - 30s - loss: 1.0912 - acc: 0.6489 - val_loss: 1.1027 - val_acc: 0.6347
Epoch 5/100
16098/16098 [==============================] - 26s - loss: 1.0444 - acc: 0.6996 - val_loss: 1.0495 - val_acc: 0.6926
Epoch 6/100
16098/16098 [==============================] - 24s - loss: 1.0541 - acc: 0.6891 - val_loss: 1.3584 - val_acc: 0.3795
Epoch 7/100
16098/16098 [==============================] - 25s - loss: 1.0672 - acc: 0.6759 - val_loss: 1.0491 - val_acc: 0.6896
Epoch 8/100
16098/16098 [==============================] - 24s - loss: 1.0386 - acc: 0.7047 - val_loss: 1.0606 - val_acc: 0.6814
Epoch 9/100
16098/16098 [==============================] - 25s - loss: 1.0425 - acc: 0.7020 - val_loss: 1.0986 - val_acc: 0.6419 - ET
A: 13s - loss: 1.0125 - acc: 0.7322
Epoch 10/100
16098/16098 [==============================] - 26s - loss: 1.0267 - acc: 0.7156 - val_loss: 1.0470 - val_acc: 0.6956 - ET
A: 1s - loss: 1.0281 - acc: 0.7141
Epoch 11/100
16098/16098 [==============================] - 25s - loss: 0.9901 - acc: 0.7528 - val_loss: 1.0388 - val_acc: 0.7020 - ET
A: 12s - loss: 0.9878 - acc: 0.7553
Epoch 12/100
16098/16098 [==============================] - 25s - loss: 1.0029 - acc: 0.7405 - val_loss: 1.0349 - val_acc: 0.7055
Epoch 13/100
16098/16098 [==============================] - 24s - loss: 0.9755 - acc: 0.7672 - val_loss: 1.0301 - val_acc: 0.7110
Epoch 14/100
16098/16098 [==============================] - 24s - loss: 0.9542 - acc: 0.7882 - val_loss: 1.0268 - val_acc: 0.7135 - ET
A: 2s - loss: 0.9540 - acc: 0.7881
Epoch 15/100
16098/16098 [==============================] - 24s - loss: 0.9476 - acc: 0.7955 - val_loss: 1.0226 - val_acc: 0.7165 - ET
A: 14s - loss: 0.9436 - acc: 0.8002
Epoch 16/100
16098/16098 [==============================] - 25s - loss: 0.9479 - acc: 0.7946 - val_loss: 1.0689 - val_acc: 0.6720 - ET
A: 8s - loss: 0.9476 - acc: 0.7952
Epoch 17/100
16098/16098 [==============================] - 24s - loss: 0.9621 - acc: 0.7813 - val_loss: 1.0190 - val_acc: 0.7184
Epoch 18/100
16098/16098 [==============================] - 23s - loss: 0.9309 - acc: 0.8120 - val_loss: 1.0050 - val_acc: 0.7353
Epoch 19/100
```

Environment Setup and testing

1. For loading the environment to train the network or do inference, we are providing the docker container with the environment setup. Below are instructions for setting in up:
    a. Pull the docker image from https://hub.docker.com/r/superbug/bug_classifier/
    b. Install nvidia-docker from https://github.com/NVIDIA/nvidia-docker
    c. Get inside the pulled container using this command:
        i. sudo nvidia-docker run -ti -p 8888:8888 bug_classifier bash
        ii. Inside the container, run $source active dl_env to activate python environment
        iii. jupyter notebook --ip 0.0.0.0 --no-browser --allow-root (this will give you the URL to open in browsere on Host)
        iv. Open the browser in host and copy the URL to see the notebook
        v. You can navigate to superbug_classifier.ipynb notebook to look in the source code

2. We are also packaging the trained model files (model.h5) and source code (superbug_classifier.ipynb) which can be opened in jupyter notebook

3. We also have inference engine in super-bug.com/demo.html where there are some example bug description files that you can paste in Bug window and try to see if classifies it right or not. You can also pull some bug reports from any of the modules in table below from https://bugzilla.mozilla.org/report.cgi and paste the summary from the tabular report in “Enter the bug” section in super-bug.com/demo.html to test the accuracy of model.

Label	Module
0	General
1	Preferences
2	Theme
3	Toolbars and Customization

Future Work

Improvement in model accuracy

1. Try word2vec embeddings instead of Glove embeddings
2. Try Keras Gensim model to generate specific skip-gram model for the dataset so that it generates custom embeddings for the data set.
3. Look at more hyper-parameter tuning to sweep on all settings and pick the best performing model in terms of accuracy
4.

Improving Integration with Alexa

1. Think about more cases in bug management system where voice control may be useful.

Appendix

Website

www.super-bug.com

Docker container for testing the software

https://hub.docker.com/r/superbug/bug_classifier/
Files available inside $~/superbug once indie the container

Dataset source

https://bugzilla.mozilla.org/report.cgi

Word embeddings

http://nlp.stanford.edu/data/glove.6B.zip