CLASSIFICATION OF CLINICAL NARRATIVES USING CONVOLUTIONAL NEURAL NETWORK

by

NIKIT RAJIV LONARI

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfilment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

DECEMBER 2018

To my parents Rajiv Lonari and Smita Lonari

Acknowledgements

I would like to acknowledge my supervisor Mr. Manfred Huber who has been great teacher during my masters. I was privileged to have a support and guidance, much freedom for expression and an opportunity to implement my ideas that would be significant towards my professional and personal growth. Additionally, I would like to acknowledge and thank my committee members, Dr. Zaruba and Dr. Levine. Furthermore, I would like to extend my sincere thanks to Dr. Darin Brezeale who introduced me to the idea behind Artificial Intelligence.

I'd also like to thank our administrative staff member and specifically, Dr. Sajib Datta and Ms. Sha'Londa Towns for their valuable support and services.

My hearty thanks to my family and to Disha and Vrutika for continuous support and inspiration. Last but not the least thanks to all my dear friends who constantly motivated through this journey and make it memorial.

December 4, 2018

Abstract

CLASSIFICATION OF CLINICAL NARRATIVES USING CONVOLUTIONAL NEURAL NETWORK

NIKIT RAJIV LONARI, MS

The University of Texas at Arlington, 2018

Supervising Professor: Manfred Huber

Patient safety is a key aspect for good consumer care. When an individual is hospitalized or receives medication the family wants the patient safety to be above all factors. For instance, a drug can do both either cure the disease or perhaps, give rise to an adverse event. A drug administered for an indicated condition has substantial power to reduce or cure a disease and further to prevent it from happening again in the future but at the risk of side effects. At present, there are several methods in patient safety and in particular in the area of signal detection and off-label drug usage identification that are incorporated in patient's safety. Even though these methods have relatively high accuracy, they have to be executed manually by a health professional who has to perform a case review which consumes a significant amount of time.

To address the above issue, this thesis explores a new method that can help identify whether the treatment has a lack of effect or not from medical text. This classification is based on the model's prediction probability where a convolutional neural network algorithm is used as a systems classifier. The input to the classifier are raw text (case narratives), reported by patients, physicians, or any other reporter in the form of text or phone calls to local safety offices. Based on this, the classifier outputs a class label indicating predicted effectiveness. Currently, the model has three layers consisting of 1-convolution, 1-relu and 1-max pooling layers. The above-mentioned model is trained and tested on 4 different medications; The average size of data is approximately between 60-120 cases for each medicine gathered from electronic health records.

This thesis is a proof-of-concept which demonstrates the automated version of an existing manual process which is carried out in many pharma companies for patient safety. Pharma companies have recently realized and begun to address the need for this transformation which will increase efficiency in patient's safety reporting to the Center of Complaint Vigilance.

Results: In this thesis, the CNN model was trained and tested on 320 and 160 clinical texts achieving an average accuracy (4 medications) of 87% and 85.76% on training and test data, respectively. Furthermore, precision of 90.3%, recall of 86.8% and F1-measure of 84.5% were achieved. In addition, this thesis also depicts a comparison between GPU (GTX-1080 hybrid) and CPU training time after running the model for 1000 epochs.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction and Background

Health professionals spend an increasingly longer period of time to examine a seemingly never ending list of clinical notes with a perspective to recognize key problems and getting an overall intuition of the patient’s health status. Particularly for complex cases this leads to information overload, delays or missing information [1]. Clinical notes mainly contain concomitant condition, drug-disease relation, adverse events, drug dosage and frequency, dates that are reported by pharmacist, physician or patient. For example, “the patient's height and weight were not reported. The patient's concurrent conditions included psoriasis” and extends with “the patient also reports that the drug effect lasts only for 12 weeks and then fails”. Reviewing these narratives, suppose for 100 cases, would easily add up to one or two hours of work for health professionals, potentially leading to an exponential growth in time consumption of physicians as the number of cases increases. According to a recent study conducted by Food, Drug and Administration (FDA) there has been inevitable growth in the number of reports concerning a drug being ineffective [3].

1.2 Motivation of the thesis

Although many industries are concerned about artificial intelligence eliminating jobs, in a highly skilled and resource-pressed environment like the one found in life science, quality, safety or regulatory affairs, time is of the essence and AI’s role is to free up life science teams to focus on what is important [2]. Given that the web never sleeps, another advantage of AI-based web and social monitoring is that it keeps working continuously- outside of office hours, so there is much less danger of falling behind [2]. What was previously an unachievable task for humans has now become viable due to computationally efficient machines.

1.3 Goal behind the thesis

The goal behind this work is to evaluate how machine learning can be applied to clinical text and further classify drug-related messages into two categories: lack of effect or not lack of effect. Incorporating machine learning to identify lack of effect in unstructured text notes can substantially reduce health professionals reviewing time. This saved time can aid health professionals focus on their work and be productive. The method used in this proof of concept is convolutional neural networks (CNN), which is state-of-the-art and has shown successful results in other domains. Convolutional neural nets have outperformed several traditional machine learning algorithms that were once the preferred choice. One of the significant features of CNN is its ability to extract and learn complex features where traditional machine learning approaches lack.

1.4 Organization of thesis

This thesis is divided into 5 chapters, the first chapter is dedicated to the introduction, background, motivation and goals. Chapter 2 is about related work and terminologies that have been used throughout this thesis. Chapter 3 explains the background and environment of the dataset alongside techniques and tools utilized that can solve one of the pharmaceutical complex business problem. Chapter 4 serves as the step by step guidance to how to address the given problem. Chapter 5 will showcase some results.

CHAPTER 2

RELATED WORK

2.1 Document classification using CNN

Classification of health-related text is considered a special case of text classification. Machine learning algorithms in Natural Language Processing (NLP) have been successfully applied: e.g. Support Vector Machines and Latent Dirichlet Allocation have been used for some tasks like classification on patient record notes [5] or other documents in diseases like diabetes, showing satisfying results [6,7]. The state-of-the-art models on document classification methods are designed for neural networks. Mikolov et al. [10] introduce an approach for learning word vector representations, Word2vec, which is simple and efficient. For neural embeddings, Le et al. [8] introduce the distributed representations of paragraphs, the Doc2vec, capturing the semantics in dense vectors. Other studies on CNNs for learning high level features have also shown competitive results. Other works by Kalchbrenner et al. [9] develop Dynamic Convolutional Neural Networks for modelling sentences. This work is the first approach using such technology to do sentence-level classification of medical text.

2.2 Medical text classification

In [4], the authors apply a CNN-based approach to categorization of text fragments, at a sentence level, based on the emergent semantics extracted from a corpus of medical text. Then compare the approach with three other methods: Sentence Embeddings, Mean Word Embeddings and Word Embeddings with BOW (bag-of-word). Our results indicate that the CNN-based approach proposed in this thesis is outperforming these approaches by at least 15% in terms of accuracy in the task of classification. It is necessary to gather training data that has been identified by medical professionals. The Merck Manual2 dataset contains articles covering a variety of topics, for instance, Brain, Cancer, etc. Each of those articles are classified under a parent class representing a category of medical issues and conditions. In total, the dataset includes 26 medical categories with 4000 sentences that were chosen at random for every category extracted from the Merck articles to use as the training data and to ensure there is no imbalance across categories. The validation dataset contains 1000 sentences from each of the categories.

2.3 Named Entity Recognition (NER)

Named entity extraction is a type of information retrieval which focuses on identifying instances i.e., names of various types of entities. For example, cancer would be an instance of disease; swelling would be an instance of symptoms and so on. One of the earliest NER models was based on decision trees [5]. In this paper [5], Sekine used features such as part-of-speech tags extracted by a morphological analyzer, character based information and a specialized dictionary. This system was developed for Japanese. Another early work was done by Bikel, Schwartz and Weischedel [6]. The authors used Hidden Markov Models (HMM) to identify named entities. Primary features like bi-gram and orthographic features like word case, word shape etc. were used. Borthwick [7] in his PhD thesis used a maximum entropy (MaxEnt) algorithm.

McCallum and Li [4] developed a Conditional Random Field based algorithm to extract NER in the coNLL-2003 shared task competition. Sarawagi and Cohen [8] propose a semi Markov CRF (Conditional Random Field) algorithm for named entity extraction. Cohen and

Sarawagi [8] further extended the semi Markov model with the use of a dictionary and the notion of a similarity function. Naidu and Sekine [5] provide a wide overall survey of NER research.

CHAPTER 3

CONCEPTS AND DESIGN

3.1 Convolution and Neural Network

Convolutional neural networks work very like ordinary neural nets. They consist of neurons and learnable weights and biases. Each neuron in the network receives an input, performs a multiplication with weight followed by the optional application of a non-linear function. The main difference is that convolutional neural networks contain groups of weights that are shared across multiple inputs to form a weight pattern that is applied to many locations in the input. The overall network still expresses a single differentiable score function: in this case, a sequence of sentences as an input to a class scores at the other end.

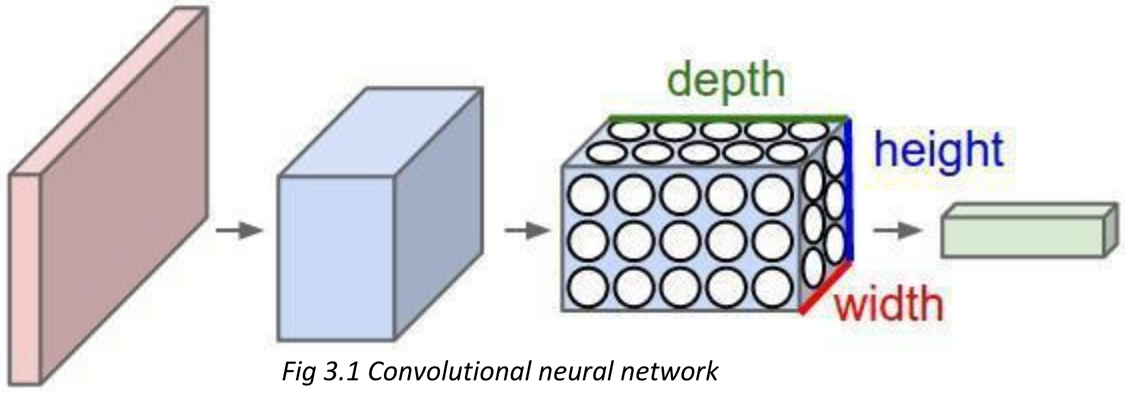

Fig 3.1 Convolutional neural network

All the layers between two ends are called hidden layer. This hidden layer, helps to transform the data and learns to extract important features that may aid to classify the given input.

3.1.1 Architecture Overview

ConvNets are sequence of layer, and each layer changes one volume of activation to another one via differentiable function. Convolutional nets consist of three main layers:

Convolutional layer, Pooling layer and Fully-Connected layer. In the final architecture, these layers are stacked on top of each other to carry out high dimensional computation.

3.1.2 Architecture Demystified

Let us begin with an example of car as an image which will be an input to the algorithm

- INPUT [32x32x3] holds the pixel values of the image with the following metrics: width 32, height 32 and 3 representing number of color channels R, G, B.
- CONV layer: this layer will take the input image and compute the output neurons that were connected to the local region of the input. Each region performs a dot product with its respective weights. This computation can result in an output with dimensionality [32x32x12] if 12 filters are used.
- RELU layer: this is most commonly used activation/transfer function in CNN, it performs elementwise transformation with the following function:

  Max (0, x)

- POOL layer: it down-samples the spatial dimensions (width, height), resulting in a reduced three-dimension matrix such as [16x16x12]. In this case, the filter size was 2 and stride of 2.
- FC layer: this layer computes scores for each class and the resulting vector has size of [1x1x10] where each of the 10 numbers depicts class probability for the given image.

In this way, ConvNets can convert/transform the original input to final class probabilities.

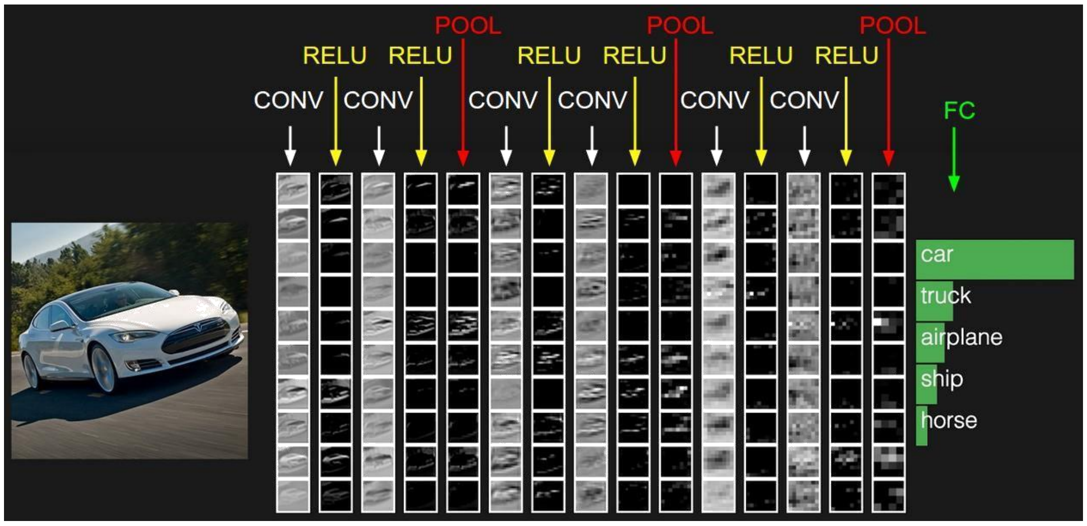

Fig 3.2 End-to-End convolutional neural network. Image downloaded from http://cs231n.github.io/convolutional-networks/

3.2 Word Embedding

Language modeling and feature learning techniques in natural language processing(NLP) together refer to word embedding. A vector of real numbers constitutes a mapping of words and phrases from the vocabulary to an input for the network. A word embedding comprises a mathematical embedding from a space with one dimension per word to a continuous vector space with potentially much lower dimensionality.

These vectors can be created with different methods like neural network, probabilistic models, dimensionality reduction on the word co-occurrence matrix, or explicit representation of words display. Whenever word and phrase embeddings are used as underlying input representation; they boost the performance of tasks like syntactic parsing and sentiment analysis in NLP.

3.2.1 Development of technique

The technique developed here mainly focuses on quantifying and categorizing semantic likeness in linguistic items depended on their distributional properties in major samples of language data. The development of modern word embedding techniques for neural networks began in 2000. Many papers were published on Neural probabilistic language models to reduce the high dimensionality of word depictions in a given context by "learning a distributed representation for words" Bengio et al [1]. How to use a "locally linear embedding" to discover depictions of high dimensional data structures was addressed by Roweis and Saul [11]. This field flourished gradually and boomed after 2010; moderately crucial advances have been made since then in terms of the enhancement of the quality of vectors and training speed of the model.

There are multiple branches and there are many research groups exploring word embeddings. The majority of new word embedding techniques are depended on neural network architectures rather than conventional n-gram models and unsupervised learning.

3.2.2 Bio-vectors

Bio vectors (BioVec) are named to refer to biological sequences and gene vectors (GeneVec) for gene sequences; this depiction can be used heavily for deep learning in proteomics and genomics in applications. The underlying patterns can be characterized using biological sequences in terms of biochemical and biophysical interpretations by BioVectors.

3.3 TensorFlow

TensorFlow is an open source software library used for numerical computation using data flow graphs. The graph has nodes which depicts mathematical operations and the graph edges depicts the multidimensional data arrays(tensors) communicated between them. It has the flexible architecture which permits computation deployed on one or more CPUs or GPUs in devices like desktop, server, or mobile devices using a single API. The researchers and engineers in Google Brain Team within Google's Machine Intelligence research organization developed tensorflow to facilitate and simlify machine learning and deep neural network research. This system is extensive and hence can be applied in a wide variety of domains [14].

A wide variety of APIs are provided by tensorflow. Complete programming control would be provided by the lowest level of the API TensorFlow Core. The TensorFlow Core is preferred by machine learning researchers and others in need of fine levels of control over respective models. TensorFlow Core is used to build easier to learn higher levels of APIs. These higher level APIs simplify the repetitive tasks and yield more consistency between different users. Tf -estimator is a high-level API which helps to manage data sets, estimators, training and inference.

3.4 Context of Dataset

Data can be collected, stored and measured. A dataset is a collection of data items where it can be analyzed in tabular form with columns representing dimensions and rows as individual records. Such a multidimensional dataset identifies each item uniquely that represents both row and dimension (it can be understood in the form of a matrix, where rows and columns are uniquely represented with their position). It can also be seen as a collection of variables and the study associated with such multivariate datasets is known as multi-dimensional data analysis or otherwise multivariate data analysis. These variables can further be grouped as sets of dependent and independent variables where dependent variables are typically described as a function of independent variables [9]. Lex [10] describes inhomogeneity or homogeneity as a fundamental property of such multivariate datasets related to data diversity. Inhomogeneity is referred to as differences in data items within the dataset and can be classified based on characteristics (refers to type), semantics (refers to meaning of the data item) and statistics (behavior or distribution of data items). In contrast, homogeneity refers to low diversity or evenly distributed data items within the dataset. Data comes in various sizes and depends on the type of system and domain it represents. In the scale of bytes, it ranges from a single byte to petabytes of data. The size of data can also be ranked as small, medium or large (big). 'Big Data' are words coined for large and complex dataset typically ranging in few gigabytes to petabytes, whereas medium sized data and small sized data range from few kilo bytes to megabytes and few bytes to kilobytes respectively. Storage systems and analytical platforms differ based on size and scale of data.

3.4.1 Understanding the data

Before passing the data as an input to an algorithm it is important to understand the data. The data used in this thesis was collected from Electronic Health Records commonly known as EHR. The data had numerous columns varying between 100-278 variables (columns) for each record. In our case it was patient records. However, our focus was to just acquire consumer complaints or case narratives. The case usually contains information like patient age, disease, drug administered, adverse event, and so on. These reports are generated by case processing teams after they receive complaints from the local safety office, physician or patient, or maybe sometimes from a pharmacist.

The model was trained and tested on datasets of size 320 and 160, respectively. In the training data, approximately 80 cases belong to one medication so, in total there were 4 medications. Each of these cases was coded as lack of effect or not a lack of effect by health professionals.

The physicians were made aware of the definition of lack of effect which states that whenever a complaint arrives and the patient explicitly reports the drug did not perform well for him then it should be coded as Lack of Effect otherwise, not. The table below show the amount of variation between two health professionals when coding the text samples.

Physician I				
Column1	**Medication1**	**Medication2**	**Medication3**	**Medication4**
LOE	107	13	56	35
Not LOE	50	107	9	45
Unlabeled	0	0	1	0
Total Case	157	120	64	80
% LOE	68.15286624	10.83333333	88.89	43.75

Fig 3.4.1 Statistics of physician I

Physician II				
Column1	**Medication1**	**Medication2**	**Medication3**	**Medication4**
LOE	105	13	52	32
Not LOE	45	104	9	45
Unlabeled	7	3	3	3
Total Case	157	59	64	80
% LOE	66.88	10.83	81.25	40

Fig 3.4.2 Statistics of physician II

The statistics above show that physicians are relatively consistent and give a clear intuition and excellent confidence that the input given to the model does not vary significantly in terms of identification.

To perform the experiments, the available data was divided into training and test data, yielding the distribution shown in Fig 3.3.

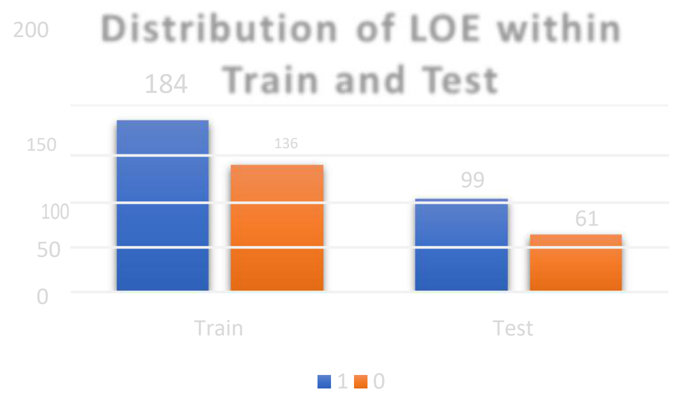

Fig 3.3 Distribution of LOE

CHAPTER 4

METHODOLOGY

The convolutional neural network implemented in this thesis works like Kim Yoon's ConvNets for Sentence Classification[13]. However, he takes advantage of a pre-trained word embedding Word2Vec. Moreover, he applied the CNN on general text data like the movie review dataset. In this thesis, convolutional neural net is applied on medical text which contains a significant amount of biomedical terminology and the model learns its own word embeddings. Furthermore, this model can achieve efficient classification performance across a range of medical text classification tasks; the architecture devised in this paper can serve as a baseline for text classification.

4.1 Data Pre-processing

Data used in the experiments is here from an Electronic Health Record database. The data set is comprised of overall 480 example complaint sentences with imbalanced ground truth. The vocabulary size of the dataset is about 8k words and the maximum length of a narrative is 740 words. Following are a few steps of data pre-processing that were applied:

1. Load lack of effect and non-LOE cases from the data files of different medications.
2. Clean the text samples by lemmatization (did not use stop word removal).
3. Stop words were not removed because in most of the cases negated words are removed which can play vital role in identification.
4. Pad each sentence with 0 to match the maximum sequence length. In this case this turned out to be 740. Padding is useful because it helps to efficiently batch the data if they are of the same length.

4.2 Network Architecture

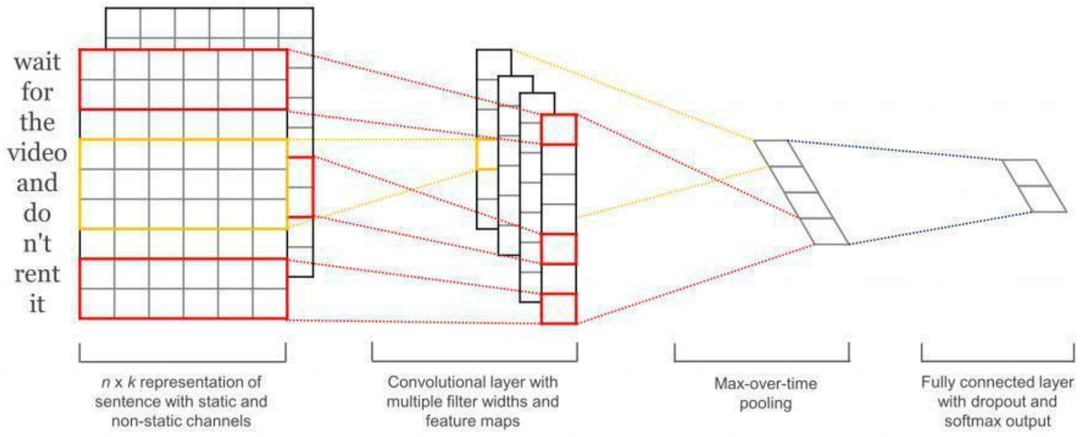
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Fig 4.1 Kim, Y (2014) Model Architecture in Understanding Convolutional Neural network for NLP from WILDML by Denny Britz. Image downloaded from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

The First layer embeds the words into low-dimensional vectors. The following layer uses multiple filters of size 3,4,5 and executes convolution over the embedded word vectors created in the previous step by sliding over multiple words at a time. Then the convolution layers output passes through relu layer. The output of the relu layer, a long feature vector, is max-pooled. We then add dropout regularization onto the max-pooling layers output, before the softmax layer is used to classify the result.

The developed model does not take advantage of any pre-trained word embedding vectors. Instead they are learned from scratch.

In the implementation in this thesis, L2 norm constraints are not enforced on the weight vectors. Convolutional Neural Networks for Sentence Classification found that the L2 norm constraints had failed to show significant effect on the result. Kim Yoon's paper experiment takes two input channels followed by static and non-static word vectors. This proof of concept only uses one channel.

4.3 Implementation

The model allows to set different hyperparameter configurations in the class constructor, and further for generating the model graph in the init function.

The following is a code snippet for the network:

```
import tensorflow as tf

import numpy as np

class CNN(object):
  """
  A CNN for text classification.
  Uses an embedding layer, followed by a convolutional, max-pooling and
  softmax layer.
  """
  def __init__(
    self, sequence_length, num_classes,
    vocab_size, embedding_size,
    filter_sizes, num_filters)
```

The class gets instantiated with the following arguments:

- sequence_length – sentences length. All the sentences where padded to the length of 740 to maintain consistent input to the CNN model.
- num_classes – Number of neurons in the output layer, two in this case.
- vocab_size – Vocabulary Size. This value is necessary to define the embedding layer with size [vocabulary_size, embedding_size].
- embedding_size – Dimensionality of embeddings which can be altered.
- filter_sizes – Number of words the model sees at a time. This can be varied based on size. Each size will have num_filters. In our case, [3, 4, 5] means that there will be filters that slide over 3, 4 and 5 words respectively, for a total of 3 * num_filters filters.
- num_filters – The number of filters per filter size. In our case, 32 for each filter.

4.4 Input Placeholders

The input data that will be passed to network:

```
# Placeholders for input, output and dropout
self.input_x = tf.placeholder(tf.int32, [None, sequence_length],
name="input_x")
self.input_y = tf.placeholder(tf.float32, [None, num_classes],
name="input_y")
self.dropout_keep_prob = tf.placeholder(tf.float32,
name="dropout_keep_prob")
```

tf.placeholder creates a placeholder variable that is fed to the network when executing it at training or test time. The second argument is the shape of the input tensor. None means that the length of that dimension could be anything. The first dimension is the batch size, and using None allows the network to handle arbitrarily sized batches. The probability of keeping a neuron in the dropout layer is also an input to the network because dropout is enabled only during training. It is disabled when evaluating the model (more on that later).

4.5 Embedding Layer

The first layer is the embedding layer, which maps vocabulary word indices into low dimensional vector representations. It is essentially a lookup table that is learned from data.

```
with tf.device('/cpu:0'), tf.name_scope("embedding"):

  W = tf.Variable(

    tf.random_uniform([vocab_size,

    embedding_size], -1.0, 1.0), name="W")

  self.embedded_chars = tf.nn.embedding_lookup(W, self.input_x)

  self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -
  1)
```

A couple of new features are used here:

- tf.device("/cpu:0") forces an operation to be executed on the CPU. By default, TensorFlow will try to put the operation on the GPU if one is available, but the embedding implementation does not currently have GPU support and throws an error if placed on the GPU.
- tf.name_scope creates a new Name Scope with the name “embedding”. The scope adds all operations into a top-level node called “embedding” to get a nice hierarchy when visualizing the network in TensorBoard.

W is the embedding matrix. It is initialized using a random uniform distribution. tf.nn.embedding_lookup creates the actual embedding operation. The result of the embedding operation is a 3-dimensional tensor of shape [None, sequence_length, embedding_size].

TensorFlow's convolutional conv2d operation expects a 4-dimensional tensor with dimensions corresponding to batch, width, height and channel. The result of embedding does not contain the channel dimension, so it is added manually, leaving a layer of shape [None, sequence_length, embedding_size, 1].

4.6 Convolution and Max-Pooling Layers

Convolutional layers are ready to be built followed by max-pooling. Here filters of different sizes are used. Because each convolution produces tensors of different shapes it is necessary to iterate through them, create a layer for each of them, and then merge the results into one big feature vector.

```
pooled_outputs = []
for i, filter_size in enumerate(filter_sizes):
  with tf.name_scope("conv-maxpool-%s" % filter_size):
    # Convolution Layer
    filter_shape = [filter_size, embedding_size, 1, num_filters]
   W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1),
   name="W")
   b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
```

```
    conv = tf.nn.conv2d(
      self.embedded_chars_expanded, W,
      Strides = [1, 1, 1, 1],
      padding="VALID",
      name="conv")
    # Apply nonlinearity
    h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
    #Max-pooling over the
    outputs pooled =
    tf.nn.max_pool(h,
      ksize=[1, sequence_length - filter_size + 1, 1, 1],
      strides=[1, 1, 1, 1],
      padding='VALID',
      name="pool")
    pooled_outputs.append(pooled)
# Combine all the pooled features
num_filters_total = num_filters * len(filter_sizes)
self.h_pool = tf.concat(3, pooled_outputs)
self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])
```

Here, W is the filter matrix and h is the result of applying the nonlinearity to the convolution output. Each filter slides over the whole embedding, but varies in how many words it covers. A "VALID" padding slides the filter over a sentence without padding the edges, performing a narrow convolution that gives an output of shape [1, sequence_length - filter_size + 1, 1, 1]. Performing max-pooling over the output of a specific filter size leaves a tensor of shape [batch_size, 1, 1, num_filters]. This is essentially a feature vector, where the last dimension corresponds to features. Once all the pooled output tensors are obtained from each filter size they are combined into one long feature vector of shape [batch_size, num_filters_total]. Using -1 in tf.reshape tells TensorFlow to flatten the dimension when possible.

Referring back to Understanding Convolutional Neural Networks for NLP to get some intuition, visualizing the operations in TensorBoard may help as well (for specific filter sizes 3, 4 and 5 here). The following figure shows the visualization from TensorBoard:

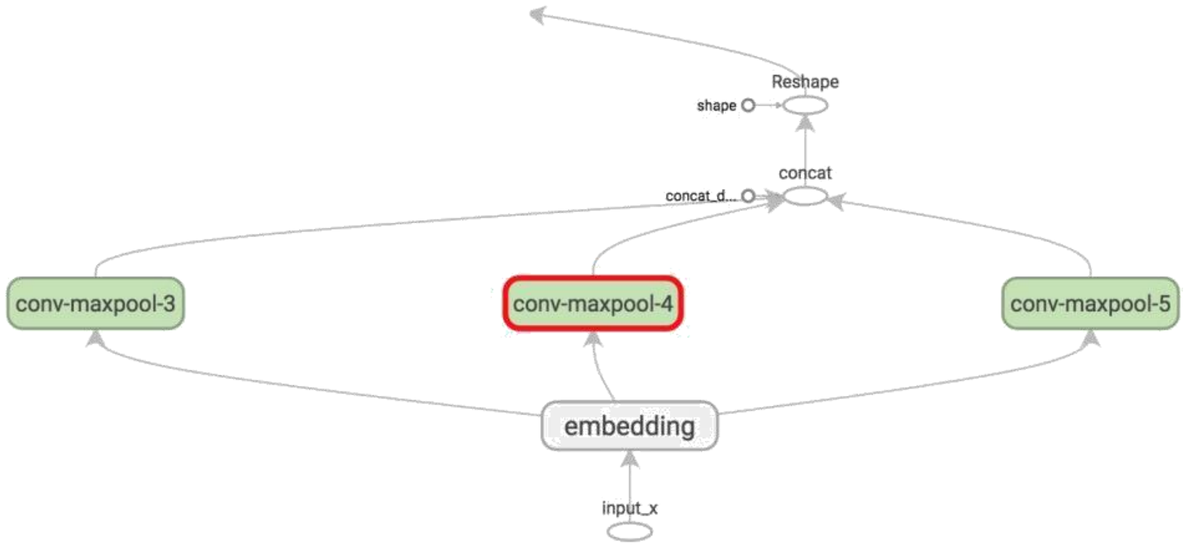

Fig 4.2 Tensorboard Graph of Convolutional neural net

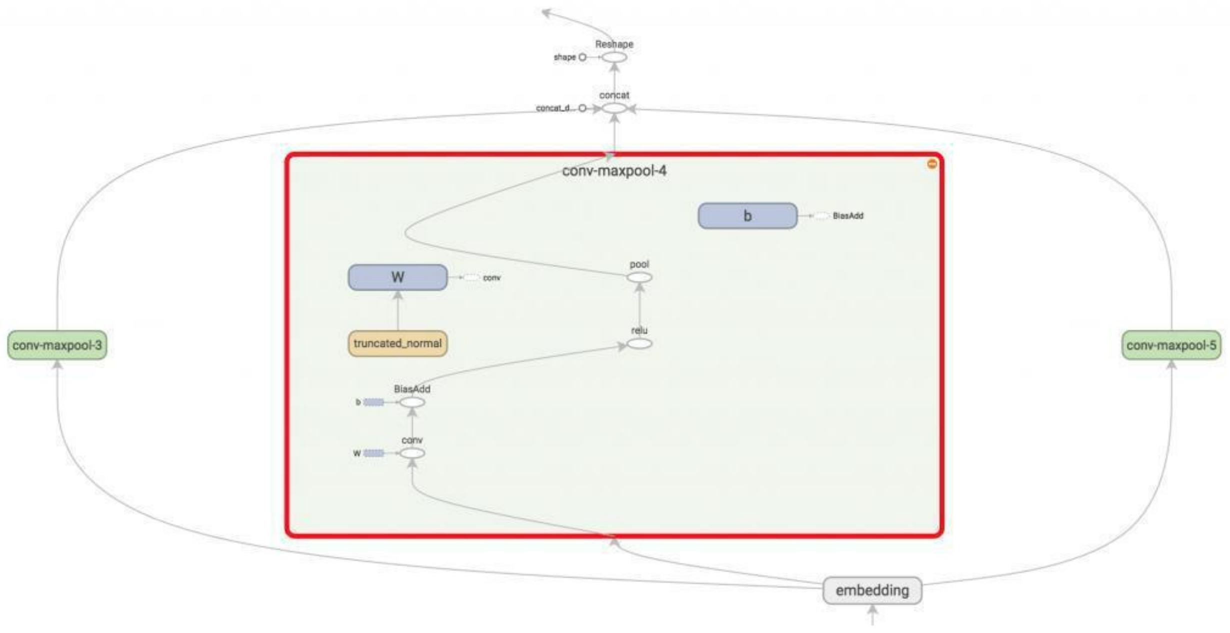

Fig. 4.3 Explored filter 4

4.7 Dropout Layer

Dropout is the perhaps most popular method to regularize convolutional neural networks. The idea behind dropout is simple. A dropout layer stochastically “disables” a fraction of its neurons. This prevents neurons from co-adapting and forces them to learn individually useful features. The fraction of neurons that is kept enabled is defined by the dropout_keep_prob input to our network. It is set to a value of 0.5 during training, and to 1 (disable dropout) during evaluation.

```
# Add dropout
with tf.name_scope("dropout"):
  self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)
```

4.8 Scores and Predictions

Using the feature vector from max-pooling (with dropout applied) predictions can be generated by doing a matrix multiplication and picking the class with the highest score. A softmax function can also be applied to convert raw scores into normalized probabilities, but that would not change the final predictions.

```
with tf.name_scope("output"):
  W = tf.Variable(tf.truncated_normal([num_filters_total,
      num_classes], stddev=0.1), name="W")
  b = tf.Variable(tf.constant(0.1,
  shape=[num_classes]), name="b") self.scores
  = tf.nn.xw_plus_b(self.h_drop, W, b,
  name="scores") self.predictions =
  tf.argmax(self.scores, 1, name="predictions")
```

Here, tf.nn.xw_plus_b is a convenience wrapper to perform the $Wx + b$ matrix multiplication.

4.9 Loss and Accuracy

Using the scores, the loss function can be defined. The loss is a measurement of the error the network makes, and the goal is to minimize it. The standard loss function for categorization problems it the cross-entropy loss.

```
#Calculate mean cross-

entropy loss with

tf.name_scope("loss"):

  losses =

  tf.nn.softmax_cross_entropy_with_logits(self.scores,

  self.input_y) self.loss = tf.reduce_mean(losses)
```

Here, tf.nn.softmax_cross_entropy_with_logits is a convenience function that calculates the cross-entropy loss for each class, given scores and the correct input labels. Then mean of the losses is taken. Sum could also be used, but that makes it harder to compare the loss across different batch sizes and train/dev data.

An expression for the accuracy is defined, which is a useful quantity to keep track of during training and testing.

```
# Calculate Accuracy
with tf.name_scope("accuracy"):
  correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
  self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"),
  name="accuracy")
```

Figures 4.4 and 4.5 show the loss and accuracy during training, respectively, illustrating the way the minimization of the loss function yields an optimization of the accuracy of the network.

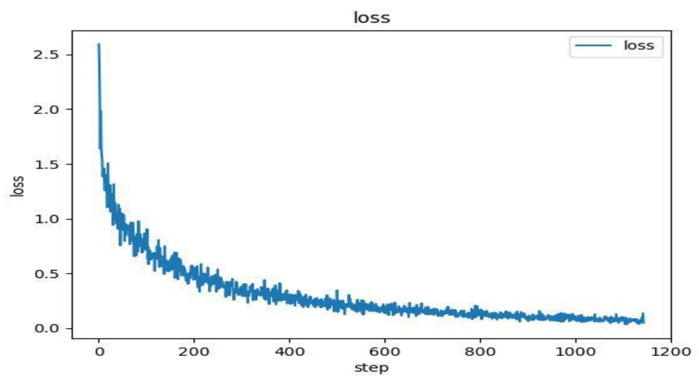

Fig. 4.4 Loss after 1000 iteration

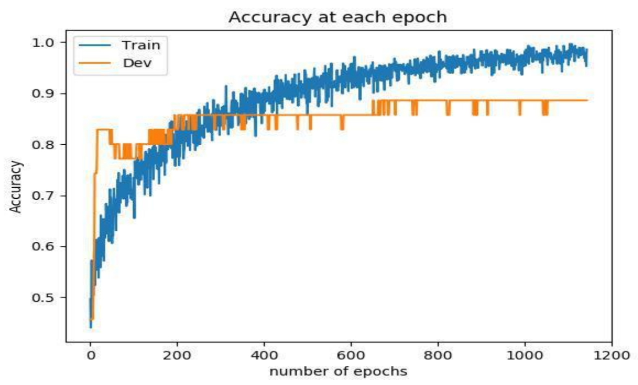

Fig. 4.5 Accuracy after 1000 iteration

4.10 Visualizing the network

Visualizing the overall network using TensorBoard yields:

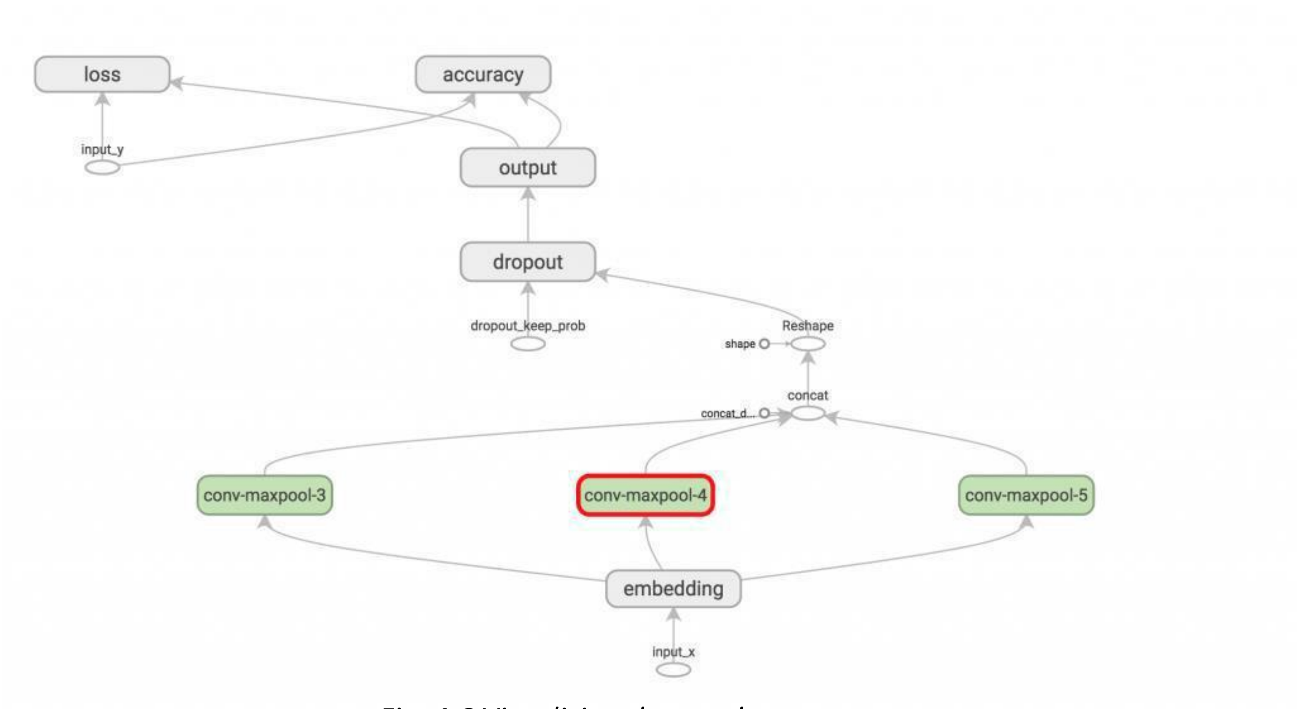

Fig. 4.6 Visualizing the graph

4.11 Training Procedure

Before training procedure is defined for the network it is necessary to understand some basics about how TensorFlow uses Sessions and Graphs.

In TensorFlow, a Session is the environment where graph operations are executed, and it contains state of Variables and queues. Each session operates on a single graph. If a session is not used explicitly when creating variables and operations then the current default session is created by TensorFlow. The default session can be changed by executing commands within a session.as_default() block (see below).

A Graph contains operations and tensors. Multiple graphs can be used in the program, but most programs only need a single graph. It is possible to use the same graph in multiple sessions, but not multiple graphs in one session. TensorFlow always creates a default graph, but a graph can be created manually and set it as the new default, like done below. Explicitly creating sessions and graphs ensures that resources are released properly when they are no longer needed.

```
with tf.Graph().as_default():
  session_conf = tf.ConfigProto(
   allow_soft_placement=FLAGS.allow_soft_
  placement,
  log_device_placement=FLAGS.log_device_p
  lacement)                  sess                  =
  tf.Session(config=session_conf)
  with sess.as_default():
    # Code that operates on the default graph and session comes here...
```

The allow_soft_placement setting allows TensorFlow to fall back on a device with a certain operation implemented when the preferred device does not exist. For example, if the code places an operation on a GPU and then runs the code on a machine without GPU, not using allow_soft_placement would result in an error. If log_device_placement is set, TensorFlow logs which devices (CPU or GPU) it places operations on. That is useful for debugging. FLAGS are command-line arguments to the program.

4.12 Instantiating the CNN and minimizing the loss

When TextCNN models are instantiated all the variables and operations defined will be placed into the default graph and session that is created above.

```
cnn = TextCNN(
  sequence_length=x_train.shape[1],
  num_classes=2,
  vocab_size=len(vocabulary),
  embedding_size=FLAGS.embedding_dim,
  filter_sizes=map(int, FLAGS.filter_sizes.split(",")),
  num_filters=FLAGS.num_filters)
```

Next, it is defined how to optimize the network's loss function. TensorFlow has several built-in optimizers. Adam optimizer is used here.

```
global_step = tf.Variable(0, name="global_step", trainable=False)

optimizer = tf.train.AdamOptimizer(1e-4)

grads_and_vars = optimizer.compute_gradients(cnn.loss)

train_op = optimizer.apply_gradients(grads_and_vars,
global_step=global_step)
```

Here, train_op is a newly created operation that can be run to perform a gradient update on the parameters. Each execution of train_op is a training step. TensorFlow automatically figures out which variables are "trainable" and calculates their gradients. By defining a global_step variable and passing it to the optimizer TensorFlow is allowed to handle the counting of training steps. The global step will be automatically incremented by one every time train_op is executed.

TensorFlow has a concept of a summaries, which allow to keep track of and visualize various quantities during training and evaluation. For example, to keep track of how the loss and accuracy evolve over time. Tracking of more complex quantities can be kept, such as histograms of layer activations. Summaries are serialized objects, and they are written to disk using a SummaryWriter.

```
#Output directory for models and

summaries timestamp =

str(int(time.time()))

out_dir = os.path.abspath(os.path.join(os.path.curdir, "runs", timestamp))

print("Writing to {}\n".format(out_dir))


# Summaries for loss and accuracy

loss_summary = tf.scalar_summary("loss", cnn.loss)

acc_summary = tf.scalar_summary("accuracy", cnn.accuracy)

# Train Summaries

train_summary_op = tf.merge_summary([loss_summary, acc_summary])

train_summary_dir = os.path.join(out_dir, "summaries", "train")

train_summary_writer = tf.train.SummaryWriter(train_summary_dir,
sess.graph_def)


# Dev summaries

dev_summary_op = tf.merge_summary([loss_summary, acc_summary])

dev_summary_dir = os.path.join(out_dir, "summaries", "dev")
dev_summary_writer = tf.train.SummaryWriter(dev_summary_dir,
sess.graph_def)
```

Here, separate tracks of summaries for training and evaluation are kept. In this case these are the same quantities, but there may be quantities that need to be tracked during training only (like parameter update values). tf.merge_summary is a convenience function that merges multiple summary operations into a single operation that can be executed.

4.13 Checkpointing

Another TensorFlow feature that typically needs to be used is checkpointing – saving the parameters of the model to restore them later on. Checkpoints can be used to continue training at a later point, or to pick the best parameter setting using early stopping. Checkpoints are created using a Saver object.

```
# Checkpointing

checkpoint_dir = os.path.abspath(os.path.join(out_dir, "checkpoints"))

checkpoint_prefix = os.path.join(checkpoint_dir, "model")

#Tensorflow assumes this directory already exists so we

need to create it if not os.path.exists(checkpoint_dir):

   os.makedirs(checkpoint_dir) saver =

tf.train.Saver(tf.all_variables())
```

4.14 Initializing variables

Before model is trained it is necessary to initialize the variables in the graph:

```
sess.run(tf.initialize_all_variables())
```

The initialize_all_variables function is a convenience function running all of the initializers that are defined for the variables. Initializers of the variables can be called manually. That’s useful if embeddings are to be initialized with pre-trained values for example.

4.15 Defining a single training step

Let us now define a function for a single training step, evaluating the model on a batch of data and updating the model parameters.

```
def train_step(x_batch, y_batch):
  """
  A single training step
  """
  feed_dict     =      {
   cnn.input_x: x_batch,
   cnn.input_y: y_batch,
   cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
  }
```

```
 step, summaries, loss, accuracy = sess.run(

  [train_op, global_step, train_summary_op, cnn.loss, cnn.accuracy],

  feed_dict)

time_str = datetime.datetime.now().isoformat()

print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss,
accuracy))

train_summary_writer.add_summary(summaries, step)
```

feed_dict contains the data for the placeholder nodes to be passed to the network. Values must be fed for all placeholder nodes, or TensorFlow will throw an error. Another way to work with input data is using queues, but that is beyond the scope of this description.

Next, train_op is executed using session.run, which returns values for all the operations which are told to be evaluated. Note that train_op returns nothing, it just updates the parameters of the network. Finally, the loss and accuracy of the current training batch is printed and summaries saved to disk. Note that the loss and accuracy for a training batch may vary significantly across batches if the batch size is small. And because dropout is used, the training metrics may start out being worse than the evaluation metrics.

A similar function is written to evaluate the loss and accuracy on an arbitrary data set, such as a validation set or the whole training set. Essentially this function does the same as the above, but without the training operation. It also disables dropout.

```
def dev_step(x_batch, y_batch, writer=None):

  """

  Evaluates model on a dev set

  """

  feed_dict = {

   cnn.input_x: x_batch,

   cnn.input_y: y_batch,

   cnn.dropout_keep_prob: 1.0

  }

  step, summaries, loss, accuracy = sess.run(

    [global_step, dev_summary_op, cnn.loss, cnn.accuracy],

    feed_dict)

  time_str = datetime.datetime.now().isoformat()

  print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss,
  accuracy))

  if writer:

    writer.add_summary(summaries, step)
```

4.16 Training Loop

Finally, the training loop can be written. It iterates over batches of the data, the train_step function is called for each batch and occasionally the model is evaluated and checkpointed:

```
# Generate batches

batches = data_helpers.batch_iter(

zip(x_train, y_train), FLAGS.batch_size, FLAGS.num_epochs)

#Training loop. For each batch...

for batch in batches:

  x_batch, y_batch = zip(*batch)

  train_step(x_batch, y_batch)

  current_step = tf.train.global_step(sess, global_step)

  if current_step % FLAGS.evaluate_every == 0:

    print("\nEvaluation:")

    dev_step(x_dev, y_dev, writer=dev_summary_writer)

    print("")

  if current_step % FLAGS.checkpoint_every == 0:
    path = saver.save(sess, checkpoint_prefix, global_step=current_step)

    print ("Saved model checkpoint to {}\n".format(path))
```

Here, batch_iter is a helper function written to batch the data, and tf.train. global_step is a convenience function that returns the value of global_step.

CHAPTER 5

RESULTS

The convolutional model was trained with a minimum of 1000 epochs at a learning rate of 0.0001. As seen above in Figures 4.4 and 4.5, the model could reduce the loss and simultaneously increase the accuracy. The CNN model was trained and tested on 320 and 160 clinical texts achieving an average accuracy (4 medications) of 87% and 85.76% on training and test data, respectively. Furthermore, a precision of 90.3%, recall of 86.8% and F1-measure of 84.5% was achieved. More detailed results are shown in Table 5.1. In addition, this thesis also depicts the comparison between GPU (GTX-1080 hybrid) and CPU training time after running the model for 1000 epochs. Figure 5.1 shows the training time comparison and shows the significant advantage of GPU training.

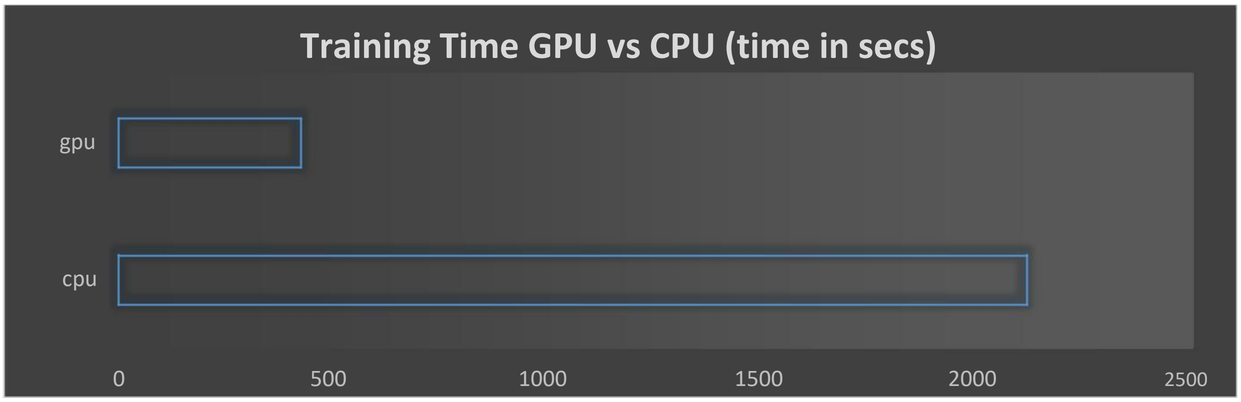

Fig. 5.1 Training time comparison

	LOE			NON-LOE		
MEDICATION	**PRECISION**	**RECALL**	**F1MEASURE**	**PRECISION**	**RECALL**	**F1-MEASURE**
MEDICATION I	0.91	0.9720	0.9412	0.9302	0.8	0.8602
MEDICATION II	0.98	0.7692	0.8619	0.9727	0.81	0.8839
MEDICATION III	.9622	0.9807	0.9714	0.875	0.7778	0.8235
MEDICATION IV	0.8444	0.76	0.8	0.4783	0.6111	0.5366

Table 5.1 Precision, Recall and F1-Measure

5.1 Comparison

This thesis also compares the accuracy results of CNN model with a Random Forest model. Before giving the input to random forest model, the data undergoes the same pre-processing steps. Following are the pre-processing steps:

- Vocabulary
- Initialize word embedding matrix for each word in the vocabulary

After these steps, the input given to the random forest model would be the top 96 features calculated by our Convolutional neural network from filters of size 3x50, 4x50 and 5x50. In the convolutional neural net the final layer is just a logistic regression with output representing the prediction probability of each class. In this thesis, there are two classes so the output layer will have 2 neurons, each will have resulting probabilities of belonging to any of the two classes. Similarly, here the final layer (logistic regression layer) is replaced with random forest decision tree. The forest consists of 50 decision tree and each of them has input bootstrapped from the top 96 features. The figure below shows accuracy metrics with their respective confidence intervals for both the models:

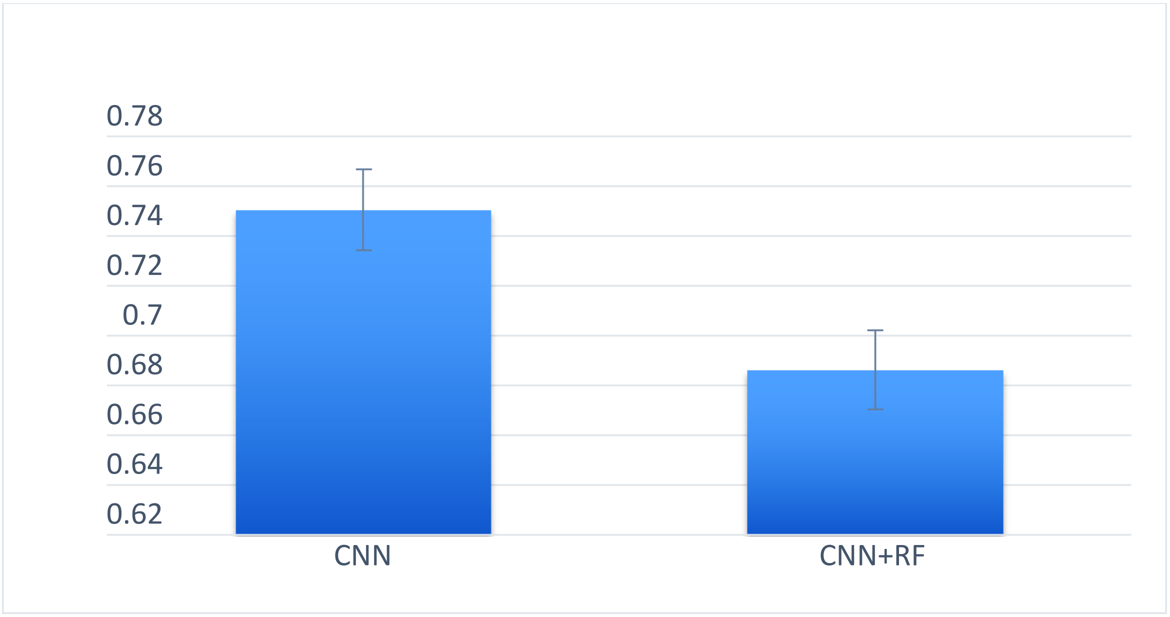

Fig 5.1 Accuracy: CNN vs CNN+RF

CHAPTER 6

CONCLUSIONS

The analysis of unstructured medical texts, including patient and nurse comments, to assess issue regarding drug effects and efficacy is an important issue to ensure patient safety and has so far to be performed largely manually by trained health care professionals. This is highly time consuming, thus leading to the question whether part of this analysis could be performed automatically to reduce the burden on the health care professional and ensure that no data is overlooked. To investigate this, this thesis develops an approach to classify medical texts “case narratives or customer complaints” using a convolutional neural network in order to predict the effectiveness of a drug. Applying the model to a dataset of case notes, the model was able to determine lack of effect with an accuracy of 87%. In a comparison study, the CNN model was shown to outperform a random forest – decision tree model by approximately 6%. This thesis also shows the benefit of using GPUs by investigating the time differences between CPU and GPU based training of the models. The GPU was equipped with 3200 CUDA cores whereas CPU had 8 cores processor.

Given the huge amounts of data the FDA receives every year from multiple sources and the resulting need for an automated system to classify lack of efficacy in order to be compliant, the baseline algorithmic approach presented here demonstrates the potential of such a system to increase the use and analysis of currently low valued submissions of LOE reports to the FDA by automatically analyzing them and thus reducing the need of person time to do so manually.

REFRENCES

[1] Y. Bengio, R. Ducharme, P. Vincent. 2003. Neural Probabilitistic Language Model. Journal of Machine Learning Research 3:1137–1155.

[2] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research 12:2493–2537.

[3] U.S Department of Health and Human Services, Food and Drug Administration, 1997. Guidance for Industry

[4] Mark Hughes, Irene Li, Spyros Kotoulas, Toyotaro Suzumura. Medical Text Classification using Convolutional Neural Network

[5] Cohen, R., Aviram, I., Elhadad, M., & Elhadad, N. (2014). Redundancy-aware topic modeling for patient record notes. PloS one, 9(2), e87555.

[6] Marafino, B. J., Davies, J. M., Bardach, N. S., Dean, M. L., Dudley, R. A., & Boscardin, J. (2014). Ngram support vector machines for scalable procedure and diagnosis classification, with applications to clinical free text data from the intensive care unit. Journal of the American Medical Informatics Association, 21(5), 871-875.

[7] Wang, L., Chu, F., & Xie, W. (2007). Accurate cancer classification using expressions of very few genes. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 4(1), 40-53

[8] Le, Q. V., & Mikolov, T. (2014, June). Distributed Representations of Sentences and Documents. In ICML (Vol. 14, pp. 1188-1196).

[9] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. arXiv preprint arXiv:1404.2188.

[10] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[11] S. Roweis and L. Saul (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding.

[12] http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[13] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751.

[14] https://www.tensorflow.org/

BIBLIOGRAPHY

NIKIT RAJIV LONARI joined the University of Texas at Arlington in Fall 2014 for MS in Computer Science. He received his Bachelors in Engineering in Computer Science and Engineering from Shivaji University in 2014. In the US, he worked as an Artificial Intelligence Intern at Johnson and Johnson. His current research interests are in the areas of data analysis or mining, machine learning, deep learning, computer vision.