CSC 591 ADBI Capstone

Comparative analysis of CNN, RNN and HAN for text classification

Tushar Dahibhate - tdahibh
Tejas Gupta - tdupta4
Shraddha Dhyade - sddhyade

Description

In this project, we have developed and compared the three different deep neural network (DNN) architectures for the task of text classification. The following three architectures should be explored:

CNN: Convolutional Neural Networks

RNN: Recurrent Neural Networks

HAN: Hierarchical Attention Networks

In addition we used the Word vectors generated by Google's GloVe which is an unsupervised learning algorithm as the underlying data model

Why Text Classification?

Text classification is one of the most important Natural Language Processing &

Supervised Machine Learning tasks in different business problems.

Example business applications include but not limited to:

- Understanding audience sentiment from social media
- Detection of spam & non-spam emails
- Auto tagging of customer queries
- Categorization of news articles into predefined topics

Choosing the data set

The Yelp Reviews CSV file dataset contains 10,000 reviews, of which we lay emphasis on the following information for each:

stars (1–5 rating for the business)

text (Review text)

The type information provided detail about the text type. Upon analysis we determined that each text type is review.

The dataset also contains the following columns, business_id, date, review_id, user_id and comments on the review, given by other users like **cool** / **useful** / **funny**

Data Preprocessing

Before performing the embedding and mapping steps, import all the libraries and datasets. Then, perform the following for the data sample:

- Data Cleaning: Find and remove the missing(na) values
- Determine the categorical values
- Removing stopwords with NLTK
- Splitting the data-set into Training, Test and Validation Sets

Word Embeddings

We performed word-vector embeddings that collects more information into fewer dimensions. These word embeddings map the statistical structure of the language used in the corpus.

We made use of Google's Glove as our underlying data model for pre-trained word embeddings.

Next, we tokenize the data into a format that can be used by the word embeddings. Keras offers a couple of convenience methods for text preprocessing and sequence preprocessing which we employed to prepare our text.

Building the Model

To define the layers of our model we'll use the Keras Model API. This is given some input and output tensors to include all the layers required in the computation of the tensors.

Keras provides a Dense layer in Keras, wherein each value in this layer will be fully connected to all value in the next layer. It requires the two parameters, the dimensionality of the layer's output and the shape of our input data. We have used the common values : a power of 2 as the number of dimensions, 512.
The number of rows in our input data, batch size and the number of columns as the size of our data vocabulary. The *relu* activation parameter then tells our model how to calculate the output of a layer.

Training and Evaluating the Model

To prepare our model for training, we called the compile method with the loss function we want to use, the type of optimizer, and the metrics our model should evaluate during training and testing.

We have used the *categorical_crossentropy* loss function, since each of our reviews can only belong to one user.

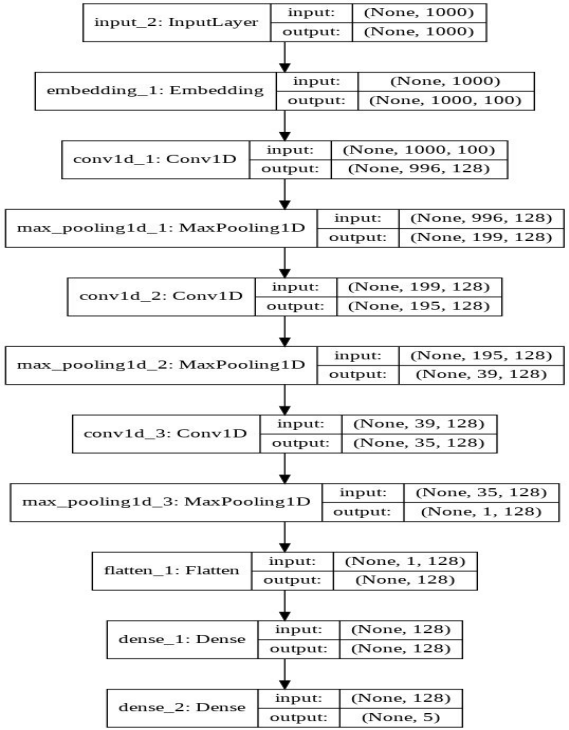
The optimizer is the function our model uses to minimize loss. We have modified rmsprop optimizer and used it as the optimizer for the model. The RMSprop optimizer is similar to the gradient descent algorithm with momentum. It restricts the oscillations in the vertical direction. This enabled us to increase our learning rate and the algorithm simultaneously took larger steps in the horizontal direction converging faster.

Training and Evaluating the Model

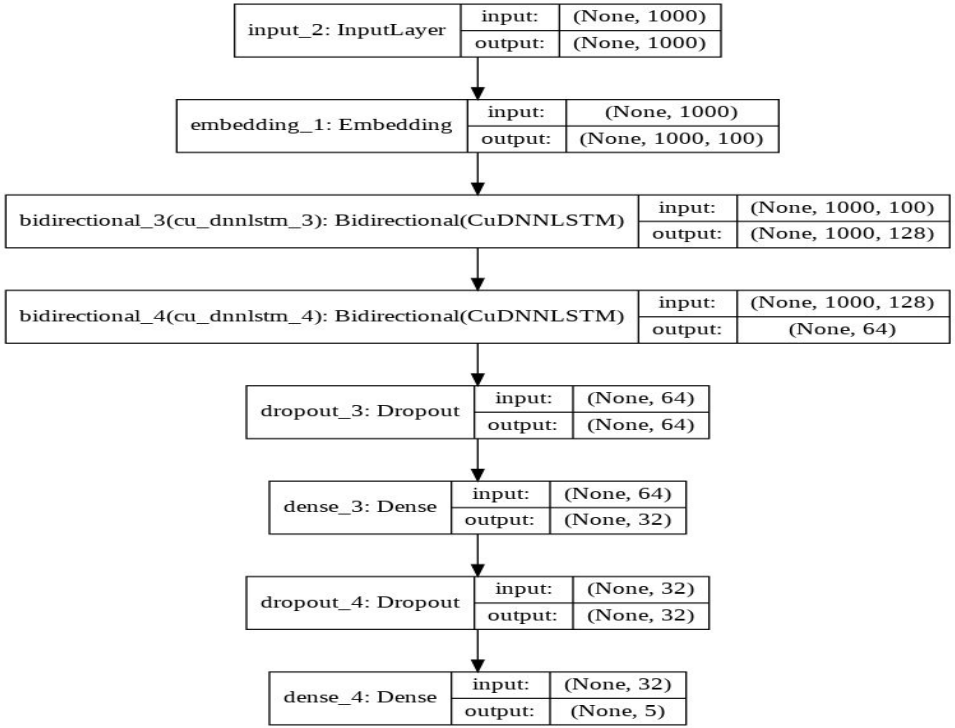
To train our model, we have called the fit() method and passed to it our training data and labels, the number of examples to process in each batch (batch size), how many times the model should train on our entire dataset (epochs), and the validation data. validation_data tells Keras the parts of training data to reserve for validation.

The goal is to generate accurate predictions on questions the model hasn't seen before. To do this we have computed our model's accuracy on our test set, which was hidden from the model during the training process using the evaluate() method and passed to it our test data and batch size, along with verbose parameter that takes in the verbosity mode (0 = silent, 1 = verbose, 2 = one log line per epoch). We have taken this verbose value to be 1.

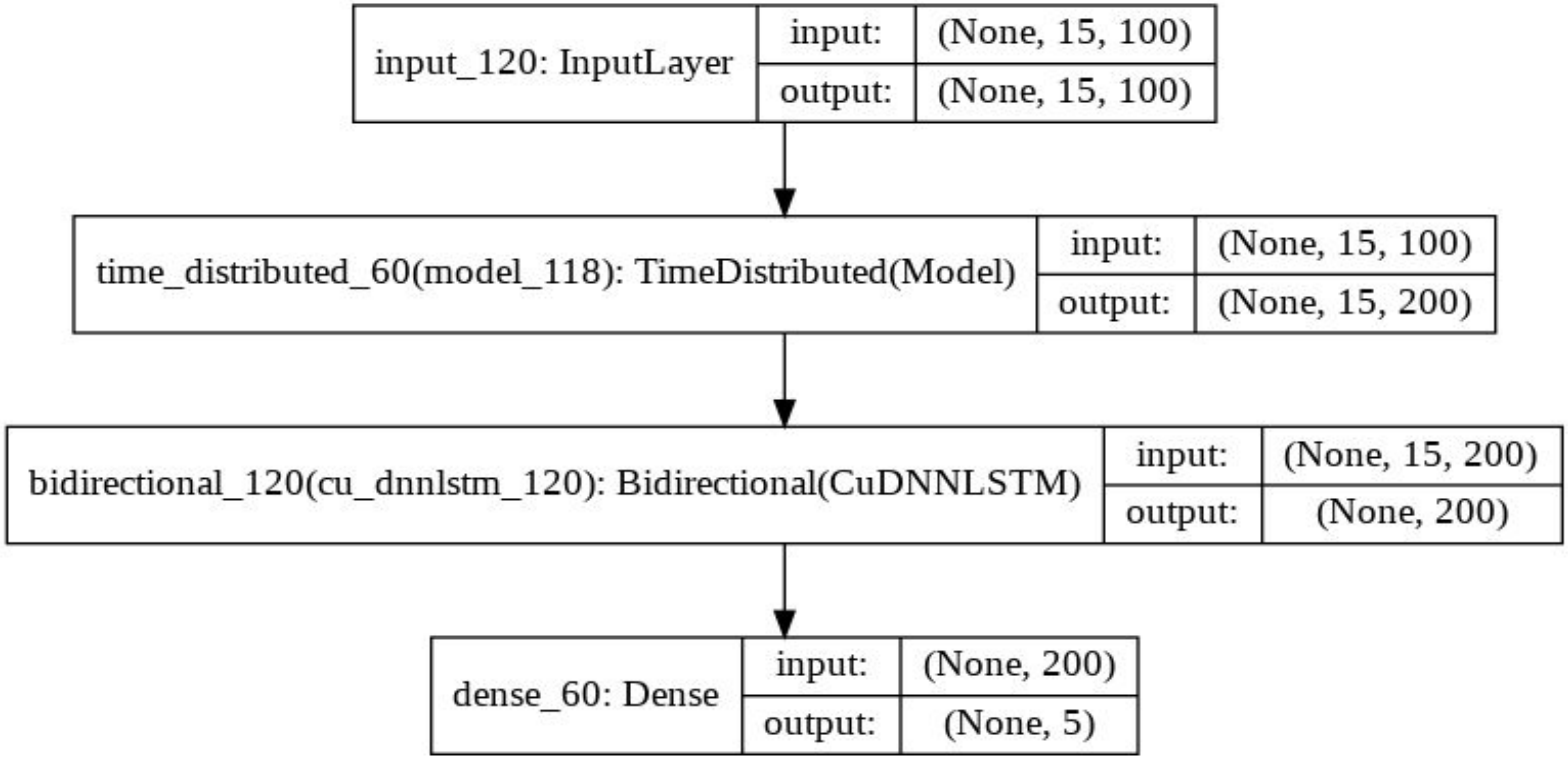
Convolutional Neural Networks



Recurrent Neural Networks



Hierarchical Attention Networks



Hyperparameter Tuning

We achieved hyperparameter tuning using grid search.

We used one-fold cross validation for this purpose.

The best parameters obtained from hyperparameter tuning were then used to train our model to get best accuracy on training and validation sets.

We tuned the model by tweaking following hyperparameters-

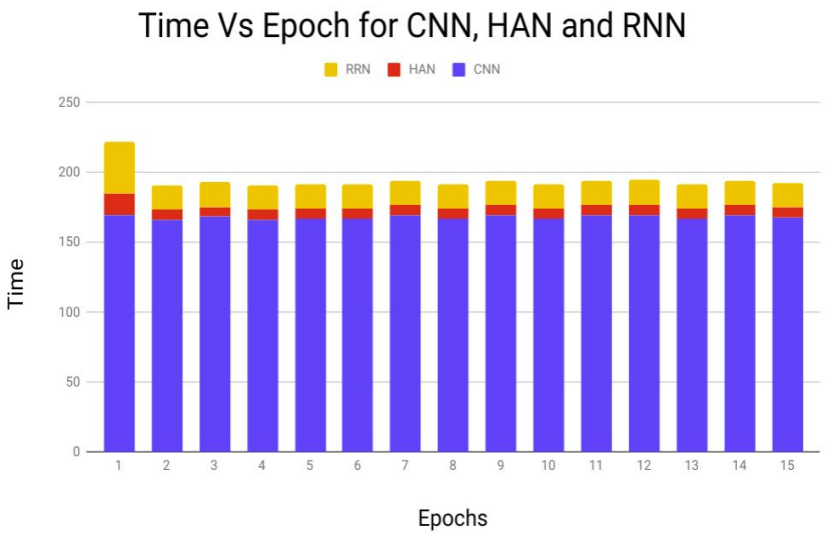
- Learning rate
- Regularization parameter
- Batch size

Results

Accuracy Results

Test accuracy of 0.9355 was achieved in HAN model, while RNN gave 0.8529 accuracy and CNN gave 0.7319 accuracy. Thus, we can say that HAN performed better among the three.

Time vs Epoch Results



Useful Resources

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

RNN: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

HAN for Text Paragraphs and Documents: https://arxiv.org/pdf/1506.01057v2.pdf

HAN for Text Classification: https://www.cs.cmu.edu/~diyiy/docs/naacl16.pdf

https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90

https://machinelearningmastery.com/cnn-long-short-term-memory-networks/

References

Tagging Posts:

https://cloud.google.com/blog/products/gcp/intro-to-text-classification-with-keras-automatically-tagging-stack-overflow-posts

Keras : https://realpython.com/python-keras-text-classification/

RMSprop Optimizer:
https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b

News classifier :
https://towardsdatascience.com/text-classification-in-keras-part-1-a-simple-reuters-news-classifier-9558d34d01d3