Fieldwork Report **Shreya Jain**
SAP Ariba Inc. Date: Aug 18, 2018

I worked as an Software Engineering Intern at SAP Ariba Inc. from May 29 to Aug 17, 2018. The company provides software based tools to enable companies to facilitate and improve the procurement process. I worked with the Search Team of the company, which make search tool whose capabilities are used by many products of the company.

Problem Statement

Whenever a user searches for certain items using our service, they expect relevant results. For this purpose, we plan on building a Neural Network which can train on the keywords of the user and other features as input to predict the category of the product as the output. This would be used as a boosting factor to get better results.

Example - When a user searches for **"item xyz"**, we want as output the code **"abcd"** which maps to the category of searched item along with its confidence to boast the search results. The output "abcd" can be understood as a hierarchical convention which can classify the searched objects. Every preceding alphabet can be understood as the superclass of the next alphabet. For instance, if 'a' represents Animals, 'b' represents Mammals, which is a clear subset of 'a'. Assuming the output is a four letter category. Each alphabet can take up to 26 values. Thus to predict each alphabet correctly the number of labels per alphabet increase drastically. Due to such high number of labels predicting the last alphabet was initially dropped.

Label Predicting	Nunber of Labels
a	26
ab	676
abc	17,576
abcd	4,56,976

Data Preprocessing

The dataset for the problem was not readily available and so we had to preprocess and join data from different sources to make it suitable for our problem. Two types of log files were pulled and merged to create a dataset that we need to train on.

Challenges faced while cleaning the noise

- Some have many other languages and mixed with English. Although embeddings are available in other languages. The data we had for other language did not have good representation on our labels.
- Some customers search for products based on some alpha numeric code which cannot be removed.
- The data is not proper English but it is broken down phrases, that make less sense when semantically trying to parse the data.

Balancing the Distribution of Classes

The data was visualized to see the distribution of classes. It was seen that some classes were dominating our data. Thus, we have picked data to balance the classes as much as possible. On observing this sort of distribution we collected more data, cleaned it and added it to the existing data.

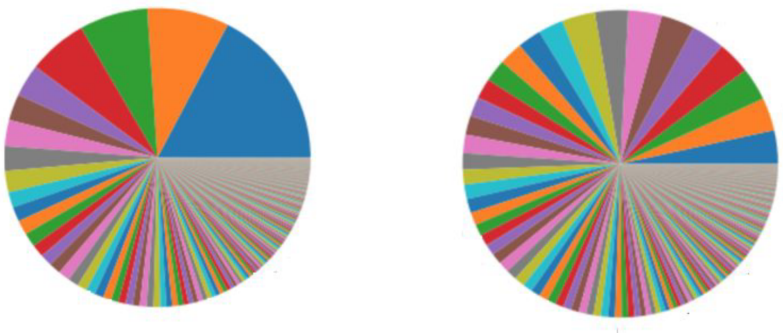

Figure 1: Class Distribution - Before balancing (left) , After Balancing (right)

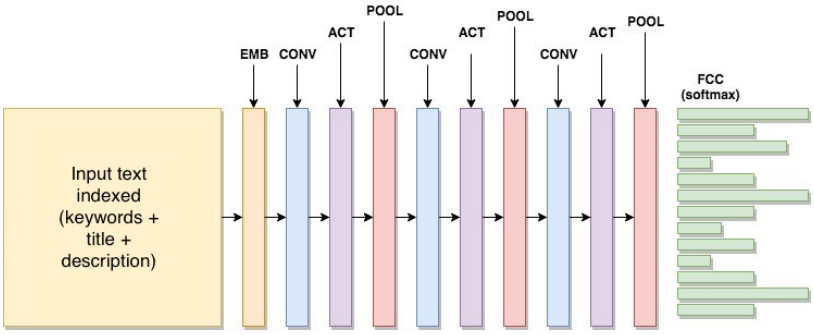

Figure 2: Model Architecture

Convolutional Neural Networks

How to apply CNN to NLP?

- Generally, CNN use image pixels as thy are applied to visual data, but when used in NLP the input is matrix of sentences or documents. Each row in this matrix is text represented as a vector. [1].
- Text can be simply represented as bag of words or word embeddings (vector representations in a multi dimensional space that help capture semantic meaning of the word) like word2vec, GLoVe, etc.

Why are we using CNN for NLP?

- CNNs are applied to text problems and it is seen that they perform well and are quite fast.
- As our data is noisy and hardly posses the sequential traits of NLP. Parts of phrases could be separated by several other words. Spatial representation like CNN is best.

Basic CNN Model

- Input : The words are the input.
- Embedding (EMB) : In majority of NLP problems embeddings are preferred instead of using sparse bag of words matrices.
- Convolution Layer (CONV) : Convolutional Layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input embeddings[1].
- Activation (ACT) : It is used to introduce some non linearity.
- Pooling layer(POOL) : It will perform a down sampling operation along the spatial dimensions.
- Fully Connected (FCC) : Fully connected layers with softmax layer will compute the probability of each class happening.
- Output : We could only fit till second last alphabet as label despite of wanting to predict the whole code.
- Dropout and Batch normalization[3] were used while training the model.
- Some hyperparameter tuning was done including concluding the optimizer suited for our data is "Adam"[2].

Experiments

The cleaned dataset was split between train, validation and test datasets. The sets were made and frozen so that all models can be compared. The train data is used for training the model, validation set is used to check the loss and accuracies after every epoch and test set is never seen by the model and is only evaluated on at the end. This is done to see a clearer picture of the real situation.

We must systematically carry out all trainings and testings to figure out the best combination of features, embeddings and model structure to get the best results.

- What should be our features for training data?
- Best way to represent our input (which embeddings to be used)
- Structure of the model
- Building optimized server based application to view it

Experiment 1 : Features

Data was split Model was trained on many training sets with different features and tested on test set that would resemble the actual data (only short text).

Feature A : Short text
Feature B : Short text combined with long text
Feature C : Short text and long text treated as two data points

Result : Training on short and long texts as two data points generalizes the best on the test set. Training on concatenation of short and long produced the worst results. Training on short text and tests on short text gives okayish accuracy and would be the conventional way of doing it. But it can be seen that this unconventional use of short and long text as two data points helps boost the accuracy on test set.
The hypothesis for this is the fact that the filters are trained to recognize longer text and thus, later it does not do a good job with just short words. When filters are trained on short and long texts as two data points, long texts gives insightful information for classifying categories and short texts helps them learn they that words can be short too (prevents from overfitting to longer texts)

Experiment 2 : Embeddings

The idea behind all of the word embeddings is to capture with them as much of the semantical, morphological, context, hierarchical and etc. information as possible[4].

Conceptnet Numberbatch (pre trained) Vs. Elmo embeddings (trained)

ConceptNet Numberbatch is one of the state-of-the-art word embeddings that represents the semantic meaning of the words. The embeddings are built using ensemble that combines data from ConceptNet, word2vec, GloVe, and others, using a variation on retrofitting and bias removal. It can used as pretrained embeddings or can be trained further to suit your data.

The Conceptnet numberbatch not only has single word embeddings but also multi-word embeddings. For example, embedding phrases like "post-it","office chair" etc. exist which very well suited for our data.

How to use Conceptnet Numberbatch ?
Convert the words to indices of those words from the pre trained embedding file. Both single word embeddings as well as multi word embeddings.
Use keras embedding layer with the indices as features.

ELMo are word representations
The embeddings have the following characteristics:

- Contextual: The representation for every word is dependent on the entire context in which it is used.
- Deep: The word embeddings are a product of combination of all the layers of bidirectional LSTM used in training it.
- Character based: Most embeddings are word based but these were character based. Such a character based embedding would be useful for out-of-vocabulary tokens unseen in training data as the network would use morphological clues to represent these unseen token more appropriately.

  ELMo representations are purely character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary tokens unseen in training.

How to use ELMo Embeddings?

- Available to us through Tensorflow hub
- It can be used as pretrained embeddings or trained again to suit our dataset

This although is tensorflow[5] it can easily be integrated in Keras as a Lambda layer.

Result
Training on numberbatch and elmo embeddings gives pretty similar accuracies but training elmo took a much longer than numberbatch making it very slow for inference time. The thought of using elmo was it would generalize well on search terms very different from the ones seen in training. But maybe using these embeddings can be explored later when more types of data is available. It was also seen that using multi word embeddings along with single word embeddings help more than using either one of them.

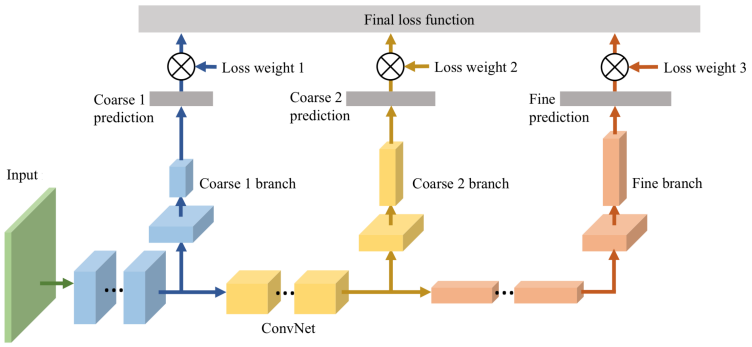

Figure 3: Branched Model Architecture

Experiment 3 : Structure of Model

CNNs for images as well as language are traditionally designed sequentially. The underlying assumption in such a design is that all target classes are equally likely. The fact that some classes are more difficult to differentiate than the others and they have an hierarchal structure of their own has led to design a model that can predict over a classification hierarchy using multiple output layers in decreasing order of class abstraction.

Branched CNN - A branched model is made that we can predict the four components (a,b,c,d) with some confidence score. This could help boost the four components of the code separately while doing the search. The structure of the model is given in fig 3[8].

Predicting the last alphabet - The number of classes for d are about 400k which would not fit in memory so we were using the Basic CNN which only predicted till the c. With the branched structure we want to use information of the predicted class and only predict the last part i.e. d (which is only 26 classes). We attached the softmax of the class prediction with features to be sent to more convolution layers. This then spit out the prediction for the last part. A matrix of product of softmax of c and d was made from which the five highest values were the full predicted code.

Loss for multiple outputs - The given loss functions are applied to all the outputs. The overall loss is minimized. The loss in case of multiple outputs is the summation of all three as default. But that being said, the weights are updated through back propagation, so each loss will affect only layers that connect the input to the loss.
Although we can use the overall loss as summation of the losses but we used **loss weights**, which are scalar coefficients to weight the loss contributions of different model outputs. The loss value that will be minimized by the model will then be the weighted sum of all individual losses, weighted by the loss weights coefficients.

Deploying the model

Flask using REST end point

It is intended for single threaded use with no concurrent requests. This method is good for deploying the model to view the inferences made for user inputs. It is slower and extremely dependent based on the version of the packages. The average time for our model per inference using a Flask app was about 100ms.

Tensorflow serving with gRPC

It is a scalable, dynamic and flexible system that can be used to serve multiple machine learning models[9]. It is designed to deploy deep learning models in production environments. TensorFlow Serving is based on gRPC, a fast remote procedure call which uses Protocol Buffers. Protocol Buffers is a serialization framework which converts objects from the memory to binary format suitable. This efficient binary format is transmitted over the network. gRPC is google's protocol call that enables remote function calls over the network. It uses Protocol Buffers to serialize and deserialize data. The average time for our model per inference using tensorflow serving and gRPC was about 20ms.

Advantages of Tensorflow serving

- Version control, lifecycle management and capability to host all kinds of ML applications.
- Inference requests are handled asynchronously in production.

- It is a lot faster than Flask as it does not have overhead like Flask and is optimized for serving tensorflow models. Also handles mini batches very well.
- It allows you update your model without stopping the server.
- Client side can be made absolutely independent of tensorflow.
- Tensorflow serving can support REST if need be.

Conclusion

During the internship, we were able to systematically understand the data and decide on the best features, embeddings and the structure of deep learning model to solve our problem the best. We are able to predict the category with about an accuracy of 87%.
The model is being used as a boasting factor. Hence even if the accuracies are a bit lower, it does not damage the search results. Moreover, on further analysis we realized that the data also had discrepancies to start with. It can be assumed that the accuracies are a bit higher than observed at first.
The model was also deployed using tensorflow serving which is stable, fast and will support multiple versions later.

Future Scope

This model helps them solve the problem for a short time. As we know the data was not readily available when we started solving the problem. A way of logging will be introduced which would store the search data. This would then help retrain the same model with better and more data as well open ways to train more complex models. With more representation of different labels, it is possible to take advantage of the hierarchical behavior of the labels.[10] First we can use a CNN for the first level of classification (predicting "abc"), then do a second level classification in the hierarchy consists of a CNN trained for the domain output in the first hierarchical level. Each second level in the network is connected to the output of the first level. For example, if the output of the first model is labeled "abc" then the network in the next hierarchical level is trained only with all subcategories of abc. So while the first hierarchical level CNN is trained with all the data, each network in the next level of the model is trained only with the data for the specified domain. This is the kind of model they plan on training as they have more data.

References

[1] Alexis Conneau, Holger Schwenk, Yann Le Cun."Very Deep Convolutional Networks for Text Classification". 15th Conference of the European Chapter of the Association for Computational Linguistics.

[2] Diederik P. Kingma, Jimmy Ba. "Adam: A Method for Stochastic Optimization". 3rd International Conference for Learning Representations, San Diego, 2015.

[3] Sergey Ioffe, Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". Print arXiv:1502.03167

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. "Distributed Representations of Words and Phrases and their Compositionality".

[5] Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". Print arXiv:1603.04467

[6] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner,Christopher Clark, Kenton Lee, Luke Zettlemoyer. "Deep contextualized word representations" NAACL 2018.

[7] Robert Speer, Joshua Chin, and Catherine Havasi (2017). "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge." In proceedings of AAAI 2017.

[8] Xinqi Zhu, Michael Bain. "B-CNN: Branch Convolutional Neural Network for Hierarchical Classification". Print arXiv:1709.09890

[9] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, Jordan Soyke. "TensorFlow-Serving: Flexible, High-Performance ML Serving". Print arXiv:1712.06139

[10] Kamran Kowsari, Donald E. Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S. Gerber, Laura E. Barnes."HDLTex: Hierarchical Deep Learning for Text Classification". ICMLA 2017.