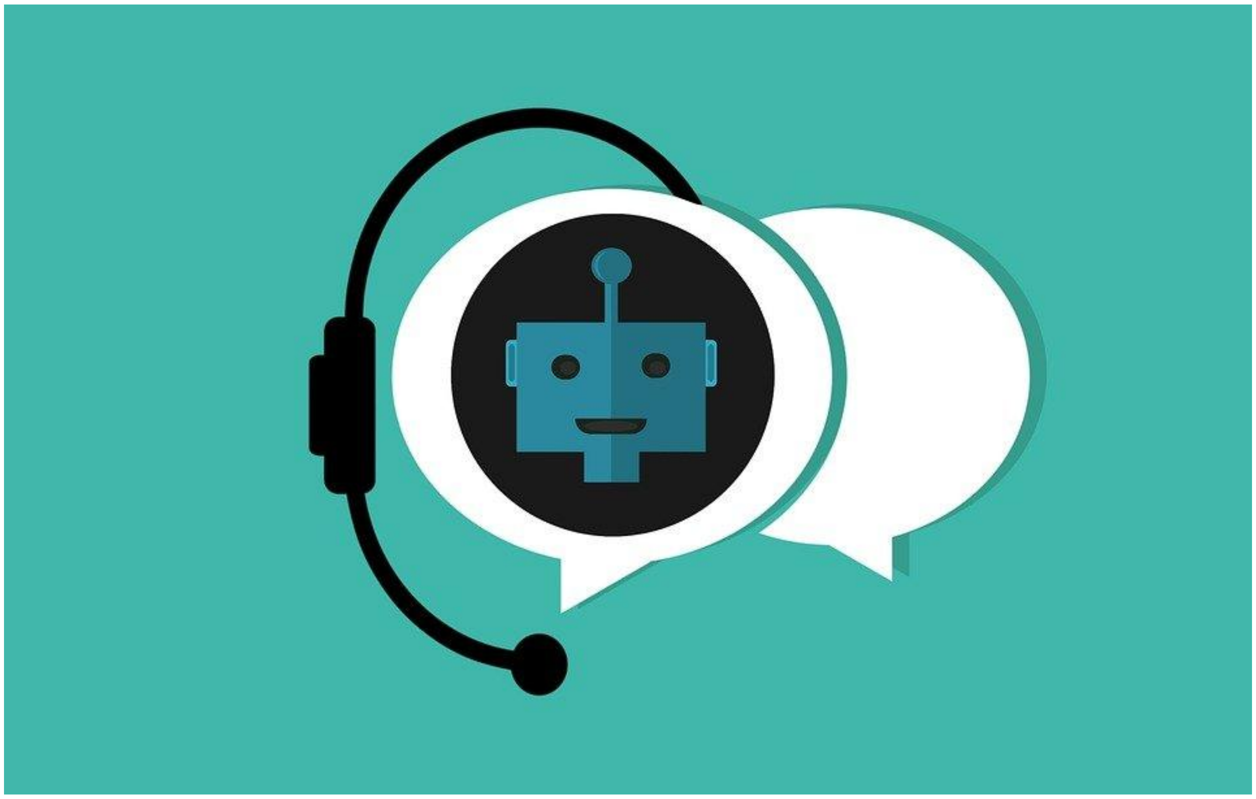

Tech_Nova: AI-Based Chatbot to reduce Operational Costs

24.12.2021

Nishant Wadhwani
Machine Learning Engineer

Overview

An enormous volume of text and voice data is generated each day. As per research, the amount of text data generated each day throughout the world is about 18 billion. How can we systemize this information to understand, summarize and analyze for useful insights ? The answer is Natural language processing - or simply NLP. Data science solutions can make an impactful difference in many ways. Technologies such as NLP can break the text data into components to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment. The machine can then decide which command to execute - based on the results of the NLP.

One of the first sectors in the market to see the application of NLP were eCommerce and retail sectors. Through in-store robots programmed with NLP-based voice recognition, Virtual avatars interacting with customers, chatbots that used analytics to pick up on customer preferences and recommend products, each one increased brand loyalty and enhanced customer experience. We’ll go through such numerous applications of NLP in this project that are trending in 2021 - and how our AI Chatbot "Tech_nova" can be helpful for eCommerce companies like Amazon or Flipkart.

Goals

1. Intelligent search functionality using Topic Modeling Techniques like LDA and NMF.
2. Sentiment Analysis using CountVectorizer, TfidfVectorizer, Word2vec,VADER SentimentIntensityAnalyzer and BERT architecture.
3. Conversational AI with Efficient Customer Support using Seq2Seq model with Attention Mechanism

Intelligent search functionality using Topic Modeling Techniques like LDA and NMF:

Specifications

Tech_Nova can provide Intelligent search which helps the employees (and also the customers) to find the information they need faster and easier. Users can use intelligent search to view any information they require, which helps them save time. Based on the interests and intent, picking from an already suggested list of options saves time. Without it, employees (and customers) would have to do things the old fashion way and search for the information they need without any help.

Intelligent search can greatly come in use to businesses that have branches with e-commerce or even if they are associated in any way. Businesses cannot use Google or other popular search engines to find answers that are business-related. For e.g. it may include questions such as why is the shipment delayed or even top customer searches for the past month. Intelligent search is always trying to give specific answers for your particular business. It is different when compared to traditional search engines since they do not show these answers. So, how does intelligent search work? The answer to that is: understanding human language. Data that businesses use is updated all the time and written in domain terminology.

Natural language processing enables intelligent search to understand and also query digital content from various data sources. Semantic search helps intelligent search to break down linguistic terms, synonyms, social text and any relations in everyday language. With this type of help, intelligent search can identify fast elements such as headers, footers, tables, or charts. It can also recognize documents like contracts or purchase orders.

We will implement this intelligent search using Topic modeling techniques like LDA(Latent Dirichlet Allocation) and NMF(Non-negative Matrix Factorization) which allows us to efficiently analyze large volumes of texts by clustering documents together into topics. A large amount of text data is unlabeled, which means that we won't be able to apply supervised learning approaches because those machine learning models would depend on historical label data. But in the real world, specifically for text data, we're not going to have a convenient label attached to a text dataset such as positive or negative or spam versus ham.

Instead, we may have a variety of labels such as the different categories that a newspaper article could be in, and we may just have the text itself unlabeled. So it's up to us to try to discover those labels through topic modeling. So, if we have unlabeled data, then we can attempt to discover these labels and in the case of text data, this means attempting to discover clusters of similar documents grouped, hopefully by some sort of topic.

A very important idea to keep in mind here is that it's very difficult to evaluate an unsupervised learning model's effectiveness because we didn't know the correct topic or the right answer, to begin with. All we know is that the documents clustered together share some sort of similar topic ideas. It's up to the user(employee) to identify what these topics represent.

Topic modeling is an unsupervised approach of recognizing or extracting the topics by detecting the patterns like clustering algorithms which divides the data into different parts. The same happens in Topic modeling in which we get to know the different topics in the document. This is done by extracting the patterns of word clusters and frequencies of words in the document.

So based on this it divides the document into different topics. As this doesn't have any outputs through which it can do this task hence it is an unsupervised learning method. This type of modeling is very much useful when there are many documents present and when we want to get to know what type of information is present in it. This takes a lot of time when done manually and this can be done easily in very little time using Topic modeling.

Dataset Details:

Our dataset will contain 10000 reviews of customers about some product like shoes, clothes, gadgets, devices, or appliances and our goal is to build a topic modeling system on this dataset to build the intelligent search, so that in future if an employee of a BU search about any product reviews, he/she can easily get some useful information regarding that product.

LDA Algorithm:

In LDA, latent indicates the hidden topics present in the data then Dirichlet is a form of distribution. Dirichlet distribution is different from the normal distribution. When ML algorithms are to be applied the data has to be normally distributed or follows Gaussian distribution. The normal distribution represents the data in real numbers format whereas the Dirichlet distribution represents the data such that the plotted data sums up to 1. It can also be said as Dirichlet distribution is a probability distribution that is sampling over a probability simplex instead of sampling from the space of real numbers as in Normal distribution.

Flow:

->We're going to imagine we have a set of documents.

->Next, for LDA to work, you as a user need to decide how many topics are going to be discovered. So even before you start LDA, you need to have some sort of intuition over how many topics.

->So, we choose a fixed number of topics for this cover, and then we're going to use LDA to learn the topic, representation of each document, and the words associated with each topic.

->Then we're going to go through each document and we're going to randomly assign each word in the document to one of the key topics. They're going to be really poor representations of topics since you essentially just assign every word a random topic.

->So now it's time to iterate over this and see how we can figure out how to fix these sorts of assignments. We're going to iterate over every word in every document to improve these topics.

->For every word in every document and each topic t, we're going to calculate the following:

We're going to calculate the proportion of words in Document D that are currently assigned to Topic T.

->Then we're also going to calculate the proportion of assignments to Topic T over all documents that come from this particular word W.

->And then we're going to reassign w a new topic where we choose topic T with a probability of topic T given document D Times probability of word W given topic T. So this is essentially the probability that topic T generated the word W.

After repeating that previous step a large number of times, we eventually reach a roughly steady state where the assignments for the topics are acceptable. These words and topics don't start changing that.

So in the end what we have is each document being assigned to a topic. And then what we can do is we can then search for the words that have the highest probability of being assigned a topic.

NMF Algorithm:-

Non-negative Matrix Factorization is an unsupervised algorithm that simultaneously performs dimensionality reduction and clustering. We can use it in conjunction with TF IDF to model topics across various documents. It is a statistical method that helps us to reduce the dimension of the input corpora. Internally, it uses the factor analysis method to give comparatively less weightage to the words that are having less coherence.

Some Important points about NMF:

1. It belongs to the family of linear algebra algorithms that are used to identify the latent or hidden structure present in the data.

2. It is represented as a non-negative matrix.

3. It can also be applied for topic modeling, where the input is the term-document matrix, typically TF-IDF normalized.

Input: Term-Document matrix, number of topics.

Output: Gives two non-negative matrices of the original n-words by k topics and those same k topics by them original documents.

In simple words, we are using linear algebra for topic modeling.

4. NMF has become so popular because of its ability to automatically extract sparse and easily interpretable factors.

Let's have an input matrix V of shape m x n. This method of topic modeling factorizes the matrix V into two matrices W and H, such that the shapes of the matrix W and H are m x k and k x n respectively.

In this method, the interpretation of different matrices are as follows:

V matrix: It represents the term-document matrix,

H matrix: Each row of matrix H is a word embedding,

W matrix: Each column of the matrix W represents the weightage of each word in each sentence i.e, semantic relation of words with each sentence.

But the main assumption that we have to keep in mind is that all the elements of the matrices W and H are positive given that all the entries of V are positive.

Milestones

For 1st Review:-

'Stunning even for the non-gamer: This soundtrack was beautiful! It paints the scenery in your mind so well I would recommend it even to people who hate the vid. game music! I have played the game Chrono Cross but out of all of the games I have ever played, it has the best music! It backs away from crude keyboarding and takes a fresher step with great guitars and soulful orchestras. It would impress anyone who cares to listen! ^_^'

Let's check Topic 6 words of LDA:-

['tracks', 'good', 'class', 'love', 'like', 'labor', 'childbirth', 'sound', 'concert', 'video', 'great', 'cd', 'music', 'album', 'dvd']

Let's check Topic 4 words of NMF:-

['kids', 'controls', 'raider', 'gameplay', 'levels', 'tomb', 'glitches', 'playing', 'buy', 'played', 'graphics', 'fun', 'games', 'play', 'game']

So, as we can see LDA topic modeling technique points out toward the topic related to awesome music/soundtrack related to video or album which is in synchronization with the review given by customer1.

Coming to NMF topic modeling technique, it points out the topic related to some kid's video game with good graphics and all, so this is also in synchronization with the review given my customer1 not directly but context is about kid's video-game.

For 5000th review:-

A wonderful book: This is the wonderfully engaging tale of how Bugliosi successfully prosecuted the Manson Family for the August 1969 murders. However, if you wish to learn a lot about the backgrounds of the "family" members, this is not the book for you. This book is from a law-enforcement point of view. If you pick up this book hoping to learn about Manson and his philosophy, you\'ll be disappointed. However, if you want to know how the players were implicated and prosecuted, there\'s no better book.'

Let's check Topic 13 words of LDA:-

THE TOP 15 WORDS FOR TOPIC #13

['people', 'don', 'characters', 'really', 'time', 'reading', 'just', 'great', 'books', 'like', 'movie', 'good', 'story', 'read', 'book']

Let's check Topic 0 words of NMF:-

THE TOP 15 WORDS FOR TOPIC #0

['thought', 'reader', 'chapter', 'understand', 'review', 'people', 'interesting', 'information', 'life', 'pages', 'recommend', 'author', 'written', 'reading', 'book']

Both the topic modeling techniques point out to some great books about an interesting incident.

Sentiment Analysis using CountVectorizer, TfidfVectorizer, Word2vec, VADER SentimentIntensityAnalyzer and BERT architecture:

Specifications

Our AI application Tech_nova implements Sentiment analysis which can provide data on customers' opinions. It is mostly analyzing and finding the emotion or intent behind a piece of text or speech or any mode of communication attached with a certain product/service. That way the company has a better insight into their products and services and as to what makes the customers buy them. Based on that, we can improve services and products. Today, sentiment analysis is one of the most popular NLP use cases to help businesses monitor brand and product sentiment in customer feedback, increasing brand loyalty and understanding customer needs.

E-commerce uses social media for monitoring customer interviews, and reviews to get feedback on their products. However, this cannot capture all the data that NLP can. It does provide a segment of it, but not all of it.

Most customers do expect a quick response from brands on social media. Having a quick response is very important for the brand since customers sometimes cannot wait. That means companies need a large number of people working just as social media managers answering these questions. This provides a huge opportunity for automation and that is why, implementation of NLP solutions that can process enormous volumes of textual data found in emails, social media posts, chats, blogs, etc. can become useful. It evades bias and spots even the smallest changes in customer behavior invisible to the human eye.

It can search for words and phrases that can analyze customers' opinions on a certain product. NLP can identify numerous emotions like happy, sad or angry, and determine whether data is positive, negative or neutral. By analyzing customers' sentimen, companies can learn more about customers to improve processes, better decision-making, enhanced customer satisfaction, understand market trends and more. This way they can improve their services and offer a more personalized experience. They can get themselvses and edge and stay ahead of their competition.

It is basically "the use of natural language processing to systematically identify, extract, quantify, and study affective states and subjective information. Generally speaking, sentiment analysis aims to determine the attitude of a speaker, writer, or any customer concerning some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event."

Dataset Details:

The amazon review dataset for electronics products was considered. The reviews given by the user to different products were considered. It contains reviews of 10000 users along with the labels positive or negative, we need to use Sentiment Analysis to train on this dataset for determining the contextual polarity and to correctly analyze the semantics behind the reviews. This model will help the new reviews to segregate into either positive or negative ones.

These will be the steps that will be followed:-

->Read in a collection of documents - a corpus

->Transform text into numerical vector data using a word embedding

->Create a classifier

->Fit/train the classifier

->Test the classifier on new data

->Evaluate performance

Word Embeddings are a method of extracting features out of text so that we can input those features into a machine learning model to work with text data. They try to preserve syntactic and semantic information.

We will be using different types of word embedding models for evaluation and will get the best one after comparing their overall performances.

Milestones

1.) Countvectorizer Word Embedding:

After applying this word embedding, we are going to use 12 types of ML classifiers for training the whole data and evaluate the performance.

a) Logistic Regression

```
        precision   recall  f1-score   support


0      0.84      0.85      0.85       997
1      0.85      0.84      0.85      1003


accuracy  85%
```

b) Naive Bayes

```
          precision   recall  f1-score   support

0      0.68      0.66      0.67       997
1      0.67      0.70      0.68      1003

accuracy 68%
```

c) KNN

```
          precision   recall  f1-score   support

0      0.72      0.61      0.66       997
1      0.67      0.77      0.71      1003

accuracy  69%
```

d) Linear SVM

```
          precision   recall  f1-score   support

0      0.82      0.83      0.83       997
1      0.83      0.82      0.83      1003

accuracy 83%
```

e) Kernel SVM

```
          precision   recall  f1-score   support
```

```
0      0.85     0.86     0.86      997
1      0.86     0.85     0.85     1003
```

accuracy 85.4%

f) Decision Tree

```
          precision    recall  f1-score   support

0      0.71     0.74     0.72      997
1      0.73     0.70     0.72     1003
```

accuracy 72%

g) RandomForest

```
          precision    recall  f1-score   support

0      0.84     0.86     0.85      997
1      0.86     0.84     0.85     1003
```

accuracy 85%

h) Adaboost

```
          precision    recall  f1-score   support

0      0.81     0.69     0.74      997
1      0.73     0.84     0.78     1003
```

accuracy 76%

i) Xgboost

```
          precision    recall  f1-score   support

0      0.75      0.83      0.79       997
1      0.82      0.73      0.77      1003

accuracy 78%
```

j) Catboost

```
          precision    recall  f1-score   support

0      0.83      0.84      0.84       997
1      0.84      0.83      0.84      1003

accuracy 84%
```

k) Gradient Boosting

```
          precision    recall  f1-score   support

0      0.76      0.85      0.80       997
1      0.83      0.74      0.78      1003

accuracy 79%
```

l) LightGBM

```
precision    recall  f1-score   support
```

```
0      0.85      0.86      0.86       997
1      0.86      0.85      0.85      1003
```

accuracy 84.6%

Kernel SVM performs the best with accuracy of 85.45% for CountVectorizer word embedding.

2.) TFIDF Vectorizer Word Embedding:

After applying this word embedding, we are going to use 12 types of ML classifiers for training the whole data and evaluate the performance.

a) Logistic Regression

```
         precision    recall  f1-score   support

0      0.86      0.87      0.86       997
1      0.87      0.86      0.86      1003
```

accuracy 86%

b) Naive Bayes

```
precision    recall  f1-score   support

0      0.66      0.66      0.66       997
1      0.66      0.66      0.66      1003
```

accuracy 66%

c) KNN

```
          precision    recall  f1-score   support

0      0.74      0.68      0.71       997
1      0.71      0.77      0.74      1003

accuracy  72%
```

d) Linear SVM

```
          precision    recall  f1-score   support

0      0.87      0.87      0.87       997
1      0.87      0.87      0.87      1003

accuracy 87%
```

e) Kernel SVM

```
          precision    recall  f1-score   support

0      0.87      0.87      0.87       997
1      0.87      0.87      0.87      1003

accuracy 87%
```

f) Decision Tree

```
          precision    recall  f1-score   support
```

```
0      0.72      0.72      0.72       997
1      0.72      0.71      0.72      1003

accuracy 72%
```

g) RandomForest

```
         precision    recall  f1-score   support

0      0.84      0.85      0.84       997
1      0.85      0.84      0.84      1003

accuracy  84%
```

h) Adaboost

```
         precision    recall  f1-score   support

0      0.73      0.86      0.79       997
1      0.83      0.69      0.75      1003

accuracy 77%
```

i) Xgboost

```
         precision    recall  f1-score   support

0      0.75      0.84      0.79       997
1      0.82      0.72      0.77      1003

accuracy 78%
```

j) Catboost

```
          precision   recall  f1-score   support

0      0.83      0.85      0.84       997
1      0.85      0.83      0.84      1003

accuracy  84%
```

k) Gradient Boosting

```
          precision   recall  f1-score   support

0      0.76      0.85      0.80       997
1      0.83      0.73      0.78      1003

accuracy  79%
```

l) LightGBM

```
          precision   recall  f1-score   support

0      0.84      0.84      0.84       997
1      0.84      0.84      0.84      1003

accuracy  84%
```

Linear and Kernel SVM performs the best with 87% accuracy for tfidf Vecorizer.

3.) Word2Vec Word Embedding with 2-Layer Neural Network:

10 epochs

```
loss: 0.0197 - acc: 0.9925 - val_loss: 3.4656 - val_acc: 0.5025
```

4.) VADER SentimentIntensityAnalyzer:

```
         precision    recall  f1-score   support


neg       0.86      0.51      0.64      5097
pos       0.64      0.91      0.75      4903


accuracy  71%
```

5.) BERT Architecture with 2 epochs and learning_rate = 2e-5

```
loss: 0.2533 - accuracy: 0.8952 - val_loss: 0.1941 - val_accuracy: 0.9250
loss: 0.1094 - accuracy: 0.9640 - val_loss: 0.1708 - val_accuracy: 0.9410
```

For 2 epochs training, it took around 1hour 20minutes to train the whole model on Tesla K80 GPU.

So, it's clear BERT Architecture on Sentiment Analysis gave the best accuracy which is around 94.1% on testing dataset out of all the word embeddings models. These are some valid reasons behind its extraordinary performance:

->BERT stands for Bidirectional Encoder Representations from Transformers. BERT is based on the Transformer architecture.

->BERT is pre-trained on a large corpus of unlabelled text including the entire Wikipedia(that's 2,500 million words!) and Book Corpus (800 million words). This pre-training step is half the magic behind BERT's success. This is because as we train a model on a large text corpus, our model starts to pick up a deeper and intimate understanding of how the language works. This knowledge is the swiss army knife that is useful for almost any NLP task.

->BERT is a "deeply bidirectional" model. Bidirectional means that BERT learns information from both the left and the right sides of a token's context during the training phase. The bidirectionality of a model is important for truly understanding the meaning of a language.

->And finally, the most impressive aspect of BERT. We can fine-tune it by adding just a couple of additional output layers to create state-of-the-art models for a variety of NLP tasks.

Conversational AI with Efficient Customer Support using Seq2Seq model with Attention Mechanism:

Specifications

Natural language processing has been studied for over 50 years. However, it began to reach its full potential and accuracy recently, which provides real value. We can see that through various ways. With interactive chatbots that can respond to customers automatically, and even voice assistants that we use in our everyday life. All this helps to improve the interaction itself between machines and humans.

A lot of companies and organizations have started to use chatbots. They have become a key asset in their customer service department. By using chatbots, companies and organizations have seen an improvement in their sales. It also helped their customers have a better experience overall.

The advantage of using these chatbots is that they are available for use 24 hours a day, 7 days a week. Chatbots can even work outside of business hours. They can address queries right away, not making the customer wait for hours and feel frustrated. It is possible for them to handle multiple requests at once and still respond to all of them. Additionally, they can learn specific terms (industry language) and answer specific questions from customers.

One of the most important things when having a business is customer feedback. The problem is that customers do not often respond to any type of survey or leave feedback (rarely even ratings). That is why conversational agents are being deployed so that they can determine customer satisfaction (or even frustration) with the services they were offered. This can indeed help to fix mistakes or flaws with any product, identify features that are not working properly or that customers are not satisfied with.

That is why a lot of companies are turning to machine learning and NLP, to get true customer feedback that is beneficial. In the end, companies depend on customer satisfaction, so their opinion matters and can help with improving business.

Conversational models are a hot topic in artificial intelligence research. Chatbots can be found in a variety of settings, including customer service applications and online helpdesks. These bots are often powered by retrieval-based models, which output predefined responses to questions of certain forms. Teaching a machine to carry out a meaningful conversation with a human in multiple domains is a research question that is far from solved. Recently, the deep learning boom has allowed for powerful generative models like Google's Neural Conversational Model, which marks a large step towards multi-domain generative conversational models. In this project, we will implement this kind of model in PyTorch which will become the heart of Tech_Nova.

Dataset Details:

->We will train a simple chatbot using movie scripts from the Cornell Movie-Dialogs Corpus.

The Cornell Movie-Dialogs Corpus is a rich dataset of movie character dialog:

220,579 conversational exchanges between 10,292 pairs of movie characters

9,035 characters from 617 movies

304,713 total utterances

This dataset is large and diverse, and there is a great variety of language formality, periods, sentiment, etc. We hope that this diversity makes our model robust to many forms of inputs and queries

Chatbot Flow:

-->Handle loading and preprocessing of Cornell Movie-Dialogs Corpus dataset

-->Implement a sequence-to-sequence model with Luong attention mechanism(s)

-->Jointly train encoder and decoder models using mini-batches

-->Implement greedy-search decoding module

-->Interact with the trained chatbot

Load and trim data:

Our order of business is to create a vocabulary and load query/response sentence pairs into memory.

Note that we are dealing with sequences of words, which do not have an implicit mapping to a discrete numerical space. Thus, we must create one by mapping each unique word that we encounter in our dataset to an index value. For this, we define a Voc class, which keeps a mapping from words to indexes, a reverse mapping of indexes to words, a count of each word, and a total word count. The class provides methods for adding a word to the vocabulary (addWord), adding all words in a sentence (addSentence), and trimming infrequently seen words (trim).

First, we must convert the Unicode strings to ASCII using unicodeToAscii. Next, we should convert all letters to lowercase and trim all non-letter characters except for basic punctuation (normalizeString). Finally, to aid in training convergence, we will filter out sentences with lengths greater than the MAX_LENGTH threshold (filterPairs). Another tactic that is beneficial to achieving faster convergence during training is trimming rarely used words out of our vocabulary. Decreasing the feature space will also soften the difficulty of the function that the model must learn to approximate. We will do this as a two-step process:

Trim words used under MIN_COUNT threshold using the voc.trim function.

Filter out pairs with trimmed words.

Prepare Data for Models:

Although we have put a great deal of effort into preparing and massaging our data into a nice vocabulary object and list of sentence pairs, our models will ultimately expect numerical torch tensors as inputs. One way to prepare the processed data for the models can be found in the seq2seq translation tutorial. In that tutorial, we use a batch size of 1, meaning that all we have to do is convert the words in our sentence pairs to their corresponding indexes from the vocabulary and feed this to the models.

However, if you're interested in speeding up training and/or would like to leverage GPU parallelization capabilities, you will need to train with mini-batches.

Using mini-batches also means that we must be mindful of the variation of sentence length in our batches. To accommodate sentences of different sizes in the same batch, we will make our batched input tensor of shape (max_length, batch_size), where sentences shorter than the max_length are zero-padded after an EOS_token.

If we simply convert our English sentences to tensors by converting words to their indexes(indexesFromSentence) and zero-pad, our tensor would have shape (batch_size, max_length), and indexing the first dimension would return a full sequence across all time-steps. However, we need to be able to index our batch along time, and across all sequences in the batch. Therefore, we transpose our input batch shape to (max_length, batch_size), so that indexing across the first dimension returns a time step across all sentences in the batch. We handle this transpose implicitly in the zero-padding function.

The inputVar function handles the process of converting sentences to tensors, ultimately creating a correctly shaped zero-padded tensor. It also returns a tensor of lengths for each of the sequences in the batch which will be passed to our decoder later.

The outputVar function performs a similar function to inputVar, but instead of returning a length tensor, it returns a binary mask tensor and a maximum target sentence length. The binary mask tensor has the same shape as the output target tensor, but every element that is a PAD_token is 0 and all others are 1. batch2TrainData simply takes a bunch of pairs and returns the input and target tensors using the aforementioned functions.

Define Models:

The brain of our chatbot is a sequence-to-sequence (seq2seq) model. The goal of a seq2seq model is to take a variable-length sequence as an input and return a variable-length sequence as an output using a fixed-sized model.

One RNN acts as an encoder, which encodes a variable-length input sequence to a fixed-length context vector. In theory, this context vector (the final hidden layer of the RNN) will contain semantic information about the query sentence that is input to the bot. The second RNN is a decoder, which takes an input word and the context vector, and returns a guess for the next word in the sequence and a hidden state to use in the next iteration.

Encoder

The encoder RNN iterates through the input sentence one token (e.g. word) at a time, at each time step outputting an “output” vector and a “hidden state” vector. The hidden state vector is then passed to the next time step, while the output vector is recorded. The encoder transforms the context it saw at each point in the sequence into a set of points in a high-dimensional space, which the decoder will use to generate a meaningful output for the given task.

At the heart of our encoder is a multi-layered Gated Recurrent Unit, invented by Cho et al. in 2014. We will use a bidirectional variant of the GRU, meaning that there are essentially two independent RNNs: one that is fed the input sequence in normal sequential order, and one that is fed the input sequence in reverse order. The outputs of each network are summed at each time step. Using a bidirectional GRU will give us the advantage of encoding both past and future contexts.

Note that an embedding layer is used to encode our word indices in an arbitrarily sized feature space. For our models, this layer will map each word to a feature space of size hidden_size. When trained, these values should encode semantic similarity between similar meaning words.

Computation Graph:

a)Convert word indexes to embeddings.

b)Pack padded batch of sequences for RNN module.

c)Forward pass through GRU.

d)Unpack padding.

e)Sum bidirectional GRU outputs.

f)Return output and final hidden state.

Decoder

The decoder RNN generates the response sentence in a token-by-token fashion. It uses the encoder’s context vectors, and internal hidden states to generate the next word in the sequence. It continues generating words until it outputs an EOS_token, representing the end of the sentence. A common problem with a vanilla seq2seq decoder is that if we rely solely on the context vector to encode the entire input sequence's meaning, it is likely that we will have information loss. This is especially the case when dealing with long input sequences, greatly limiting the capability of our decoder.

To combat this, Bahdanau et al. created an "attention mechanism" that allows the decoder to pay attention to certain parts of the input sequence, rather than using the entire fixed context at every step.

At a high level, attention is calculated using the decoder's current hidden state and the encoder's outputs. The output attention weights have the same shape as the input sequence, allowing us to multiply them by the encoder outputs, giving us a weighted sum that indicates the parts of encoder output to pay attention to.

Luong et al. improved upon Bahdanau et al.'s groundwork by creating "Global attention". The key difference is that with "Global attention", we consider all of the encoder's hidden states, as opposed to Bahdanau et al.'s "Local attention", which only considers the encoder's hidden state from the current time step. Another difference is that with "Global attention", we calculate attention weights, or energies, using the hidden state of the decoder from the current time step only. Bahdanau et al.'s attention calculation requires knowledge of the decoder's state from the previous time step. Also, Luong et al. provide various methods to calculate the attention energies between the encoder output and decoder output which are called "score functions":

Now that we have defined our attention submodule, we can implement the actual decoder model. For the decoder, we will manually feed our batch one-time step at a time. This means that our embedded word tensor and GRU output will both have shapes (1, batch_size, hidden_size).

Computation Graph:

->Get embedding of the current input word.

->Forward through unidirectional GRU.

->Calculate attention weights from the current GRU output from (2).

->Multiply attention weights to encoder outputs to get a new “weighted sum” context vector.

->Concatenate weighted context vector and GRU output using Luong eq. 5.

->Predict the next word using Luong eq. 6 (without softmax).

->Return output and final hidden state.

Single training iteration:

The train function contains the algorithm for a single training iteration (a single batch of inputs).

We will use a couple of clever tricks to aid in convergence:

The first trick is using teacher forcing. This means that at some probability, set by teacher_forcing_ratio, we use the current target word as the decoder’s next input rather than using the decoder’s current guess. This technique acts as training wheels for the decoder, aiding in more efficient training. However, teacher forcing can lead to model instability during inference, as the decoder may not have a sufficient chance to truly craft its output sequences during training. Thus, we must be mindful of how we are setting the teacher_forcing_ratio, and not be fooled by fast convergence.

The second trick that we implement is gradient clipping. This is a commonly used technique for countering the “exploding gradient” problem. In essence, by clipping or thresholding gradients to a maximum value, we prevent the gradients from growing exponentially and either overflow (NaN), or overshoot cliffs in the cost function.

Sequence of Operations:

-->Forward pass the entire input batch through the encoder.

-->Initialize decoder inputs as SOS_token and the hidden state as the encoder’s final hidden state.

-->Forward input batch sequence through decoder one-time step at a time.

-->If teacher forcing: set next decoder input as the current target; else: set next decoder input as current decoder output.

-->Calculate and accumulate loss.

-->Perform backpropagation.

-->Clip gradients.

-->Update encoder and decoder model parameters

Training iterations

It is finally time to tie the full training procedure together with the data. The trainIters function is responsible for running n_iterations of training given the passed models, optimizers, data, etc. This function is quite self-explanatory, as we have done the heavy lifting with the train function.

One thing to note is that when we save our model, we save a tarball containing the encoder and decoder state_dicts (parameters), the optimizers' state_dicts, the loss, the iteration, etc. Saving the model in this way will give us the ultimate flexibility with the checkpoint. After loading a checkpoint, we will be able to use the model parameters to run inference, or we can continue training right where we left off.

Define Evaluation:

After training a model, we want to be able to talk to the bot ourselves. First, we must define how we want the model to decode the encoded input.

Greedy decoding

Greedy decoding is the decoding method that we use during training when we are NOT using teacher forcing. In other words, for each time step, we simply choose the word from decoder_output with the highest softmax value. This decoding method is optimal on a single time-step level.

To facilitate the greedy decoding operation, we define a GreedySearchDecoder class. When run, an object of this class takes an input sequence (input_seq) of shape (input_seq length, 1), a scalar input length (input_length) tensor, and a max_length to bound the response sentence length. The input sentence is evaluated using the following computational graph:

Computation Graph:

-->Forward input through the encoder model.

-->Prepare the encoder's final hidden layer to be the first hidden input to the decoder.

-->Initialize decoder’s first input as SOS_token.

-->Initialize tensors to append decoded words to.

-->Iteratively decode one-word token at a time:

-->Forward pass through the decoder.

-->Obtain most likely word token and its softmax score.

-->Record token and score.

-->Prepare the current token to be the next decoder input.

-->Return collections of word tokens and scores.

Evaluate my text:

Now that we have our decoding method defined, we can write functions for evaluating a string input sentence. The evaluate function manages the low-level process of handling the input sentence. We first format the sentence as an input batch of word indexes with batch_size==1. We do this by converting the words of the sentence to their corresponding indexes and transposing the dimensions to prepare the tensor for our models. We also create a length tensor that contains the length of our input sentence. In this case, lengths are scalar because we are only evaluating one sentence at a time (batch_size==1). Next, we obtain the decoded response sentence tensor using our GreedySearchDecoder object (searcher). Finally, we convert the response’s indexes to words and return the list of decoded words.

evaluateInput acts as the user interface for our chatbot. When called, an input text field will spawn in which we can enter our query sentence. After typing our input sentence and pressing Enter, our text is normalized in the same way as our training data and is ultimately fed to the evaluate function to obtain a decoded output sentence. We loop this process, so we can keep chatting with our bot until we enter either “q” or “quit”.

Finally, if a sentence is entered that contains a word that is not in the vocabulary, we handle this gracefully by printing an error message and prompting the user to enter another sentence.

Run Model:

Finally, it is time to run our model!

Regardless of whether we want to train or test the chatbot model, we must initialize the individual encoder and decoder models. In the following block, we set our desired configurations, choose to start from scratch or set a checkpoint to load from, and build and initialize the models. Feel free to play with different model configurations to optimize performance.

```
model_name = 'cb_model'
attn_model = 'dot'
hidden_size = 500
encoder_n_layers = 2
decoder_n_layers = 2
dropout = 0.1
batch_size = 64
checkpoint_iter = 4000

clip = 50.0
teacher_forcing_ratio = 1.0
learning_rate = 0.0001
decoder_learning_ratio = 5.0
n_iteration = 4000
print_every = 1
save_every = 500
```

Milestones

For 4000 iterations training, the average loss has been reduced from 8.9 to 2.4. It took around 9 minutes to train the whole model on the Tesla K80 GPU.

Chatbot Conversation Sample:

> hello?

```
Bot: hello .
> where am I?
Bot: you re in a hospital .
> who are you?
Bot: i m a lawyer .
> how are you doing?
Bot: i m fine .
> are you my friend?
Bot: no .
> you're under arrest
Bot: i m trying to help you !
> i'm just kidding
Bot: i m sorry .
> where are you from?
Bot: san francisco .
> it's time for me to leave
Bot: i know .
> goodbye
Bot: goodbye .
```

Conclusion

Natural Language Processing is an emerging topic in the field of data science and companies have already started investing into research in this field. It's applications are numerous and every business out there wants to integrate it into their business somehow. NLP has risen beyond merely an interesting, high-tech whim, and will soon become an indispensable tool for all sectors in the industry. It would not just enable us to gain new insights but better understand our customers, and empower our own teams more effectively so as to become better, productive and automated.

The document demonstrates how our AI Chatbot "Tech_nova" can be helpful for eCommerce companies like Amazon or Flipkart. Topic Modeling, Sentiment Analysis and Efficient Customer support are the top three applications to any eCommerce organization which can gradually be extended into many other innovations.