Tensor Flow:

The project is about a basic form of Natural Language Processing (NLP) called Sentiment Analysis, in which we will classify a dataset review as positive or negative.

The following content explains how we will use neural networks to decide the sentiments of the user. While there are lot of tools like PyTorch, CNTK and MXN that support this task, we chose the most popular deep learning library Tensor Flow as our primary tool to implement the solution. We will also explain the benefits of choosing Tensor Flow over other tools and its limitations.

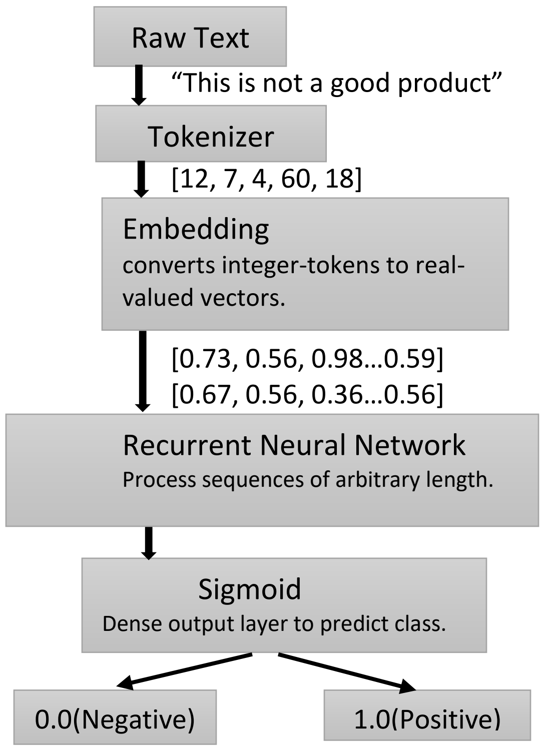
What is Tensor Flow?

TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

How Tensor Flow is used in sentiment analysis? (Contributed by Hasitha Gajjala)

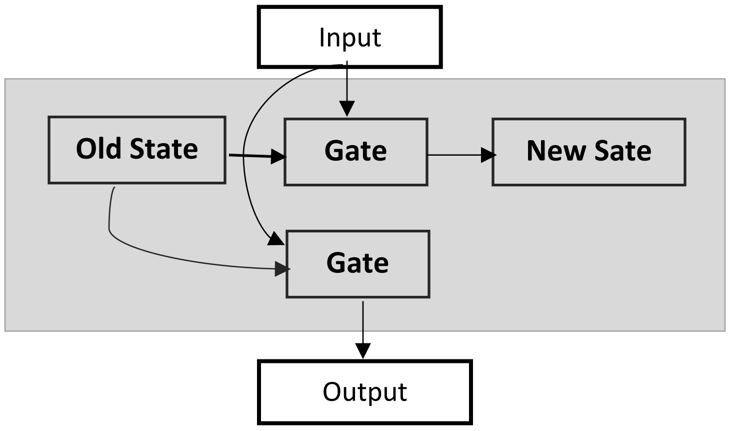
Example:

Let's start with a simple example of text "This is a not a very good product". This is a raw text entered by a user. What we want to do with this raw text is to ultimately see how it will end up as a positive sentiment or negative sentiment. This is a simple example, but in real life we have sentiments with variable length.


Raw Text
"This is not a good product"
Tokenizer
[12, 7, 4, 60, 18]
Embedding
converts integer-tokens to real-valued vectors.
[0.73, 0.56, 0.98…0.59]
[0.67, 0.56, 0.36…0.56]
Recurrent Neural Network
Process sequences of arbitrary length.
Sigmoid
Dense output layer to predict class.
0.0(Negative)
1.0(Positive)

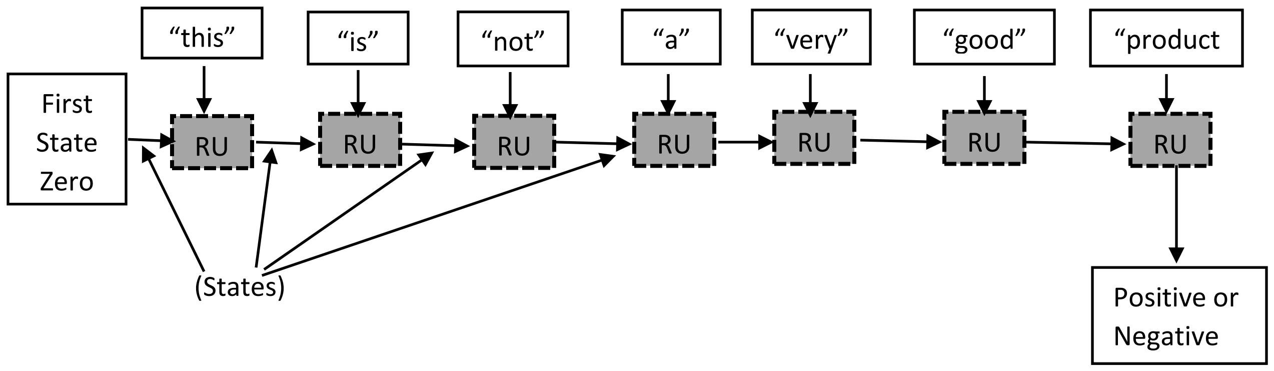
If you observe the text “good” is a positive word, but it is preceded by “not” which makes a negative sentiment. A neural network cannot work on raw text like this, we need to first convert the text into integer like tokens. We go through the entire corpus and count the number of times each word appeared and make a vocabulary. Each word gets an index into this vocabulary. So, the text now becomes [12, 7, 17, 4, 60, 18] which number corresponds to the sequence “This”, “is”, “a”, “very”, “good”, “product” respectively. A neural network still cannot understand these tokens. Suppose we have a text of 5000 words, the tokens can take any value from 0 to 5000 and they may not relate at all. Integer 1000 may have completely different semantic meaning with integer 1001. To accommodate this issue, we have an embedding layer that converts integer-tokens to real-valued vectors. The value 12 for “This” in the above text now becomes [0.73, 0.56, 0.98…0.59]. Next token 7 for “is” now becomes [0.67, 0.56, 0.36…0.56]. The mapping between tokens and embedding vectors learns the semantic meaning of the words so that words that have similar meaning are somewhat close together in this embedding space. So, now the raw text is converted into two-dimensional matrix or tensor that can now be input to what is called recurrent neural network which can take a sequence of arbitrary length. The output of this network is then fed into a fully connected or dense layer with a sigmoid activation function that outputs a value between 0 and 1. 0 is taken to be a negative sentiment and a value of 1 is meant to be a positive sentiment and a threshold, for example 0.5 in the middle. A value less than 0.5 are taken to be negative sentiment and a value greater than 1 is taken as a positive sentiment.

Recurrent Neural Network:



The basic building block of a recurrent neural network is a recurrent unit. The picture above is an example of what goes on inside a recurrent unit. The input vector is a word and it has a memory state also called old state in the picture above. Depending on the input vector and the content in memory state a new state will be produced and updated. For example, let’s say we have seen a word “good” (it will be converted to a word embedding vector) and also seen a word “not” previously, then the old State or memory state will have this word “not” and new state will be updated with “not good” which means that the whole input statement could have some negative sentiment. The mapping from the input and the old state/memory state is done through something called Gate. It is basically a matrix operation with an activation function. We have a similar gate for producing the output. Again, the output depends on the current content and the input we are seeing.

A problem with the activation function in the gate is back propagation of gradients. We need to decide in a special way, so that the gradients are not distorted too much:



The above flowchart shows what happens at different timestamps.

At timestamp t1, input vector for "this" is sent to the recurrent unit. It has its first internal memory state initialized to 0 and this is done by keras of tensor flow whenever we start processing a new sequence of data. We see the word "this" and the recurrent unit state is 0, so we use the internal gate to update the memory state and this is then used in step # 2.

At timestamp t2, we input the word "is" and now memory state has some content and there is not a lot of meaning in word "this" and the state might be still around 0. The same is with "is" which does not hold a lot of meaning and the state would still be 0 at next stamp.

At time t3, we input the word "not" which has some meaning that we ultimately want to predict as the sentiment of whole input text. This is something we need to store in the memory. So, the gate inside the recurrent unit sees that the state already contains near zero values, but now probably has some non-zero value because of "not" and saves it value.

Moving on the text timestamp t4, where we have the word "a" which also does not have much information, so it probably just ignores and copies over the previous state.

At t5, now we have the word "very" and this indicates that whatever sentiment we have seen so far, it might be a strong sentiment. So, the recurrent unit now knows that we have seen the word "not" and also seen the word "very" and it somehow stores this information in the memory state.

At next timestamp t6, we see the word "good" and all together seen so far seen "not very good" and it processes that it's a negative sentiment and stores the value in the internal state in some encoding in floating point values.

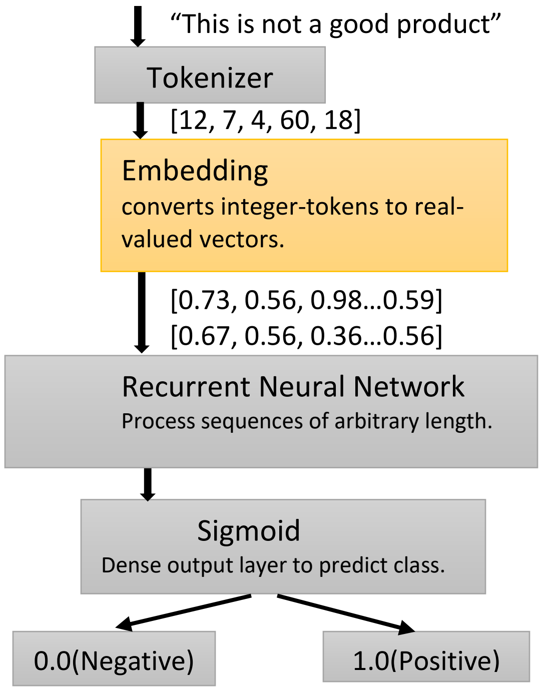
At final timestamp t7, it sees the word "movie" and this is not relevant and probably just ignores.

We then use the other gate inside the recurrent unit to output the contents of the memory state and then processed with a sigmoid function and outputs a value between 0 and 1. The idea is then to train this network on many examples of review database as possible, where for each input text we give a true sentiment value of either positive or negative and then we want tensor flow to find out what the gates inside the neural network should be so they accurately map the input text to the correct sentiment.

Exploding and Vanishing gradients in Recurrent Neural network:

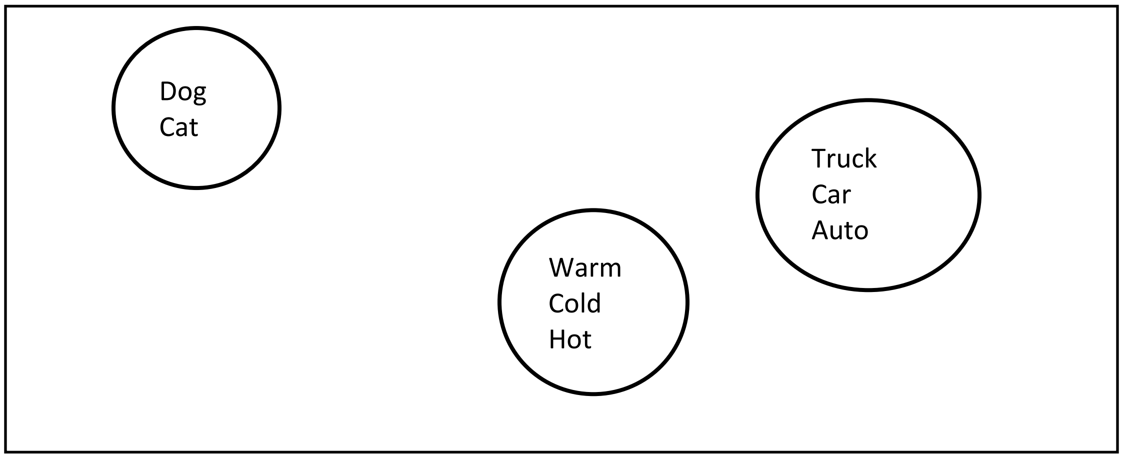
There is a phenomenon called "Exploding and Vanishing gradients which is very important in neural network. Lets' say that we have a text with 1000 words. At every time step, we apply the internal gates in recurrent unit in a recursive manner. So, if there are 1000 words then we will apply these gates 1000 times to update the internal memory state of the recurrent unit. Now the way that neural networks are trained is using back propagation of gradients so we have some loss function that gets the output of the neural network and then the true output that we desire for the given input text. We would like minimize this loss value, so that the actual output of the neural network corresponds to the desired output for this particular input text. So, we need to take the gradient of this loss function with respect to the weights inside these recurrent units and these weights are for the gates that are updating the internal states and outputting the value in the end. The gate is applied may be 1000 times and if this has a multiplication in it, what we essentially get is an exponential function. So, if we multiply a value with itself 1000 times and if it has a value < 1, then it will very quickly vanish to 0 and all information in the gradient is lost by the time we get back step to output. Similarly, a value slightly above 1 and multiplied with itself will quickly explode. The only value it can sustain is 0 and 1 i.e. 0 or 1 multiplied with itself 1000 times will remain the same. So, the recurrent unit is actually much more complicated than we see here. This is just an abstract idea that we want to somehow map the internal memory state and the input to have the internal memory state to output some value. In reality, we need to be very careful about the propagating the gradients backwards through these gates so we don't have exponential multiplication of many time steps.

How word Embedding works: (Contributed by Manojit Saha Sardar)


"This is not a good product"
Tokenizer
[12, 7, 4, 60, 18]
Embedding
converts integer-tokens to real-valued vectors.
[0.73, 0.56, 0.98...0.59]
[0.67, 0.56, 0.36...0.56]
Recurrent Neural Network
Process sequences of arbitrary length.
Sigmoid
Dense output layer to predict class.
0.0(Negative)
1.0(Positive)

As words are represented in vectors of real numbers. The idea is, words with similar vectors are semantically similar.

See example below:



If you observe above, words with similar semantics are grouped together. The hypothesis on which all word embedding is based is called “Distributional Hypothesis”. Words that occur in similar contexts tend to have similar meanings. As a consequence of this hypothesis, the meaning of a word can be approximated by the set of contexts in which it occurs.

Word embeddings (newer term) = Distributed semantic model = Distribution representation = Semantic vector space = Vector space model

There are two ways of approaching this problem. Both of them are very related to each other.

- Count word count/context co-occurrences.
- Predict word based on context and building the word embedding in the process .

Distributional hypothesis can be viewed using these examples of context below in which teacher appears. We highlight in blue the words that repeatedly occurs in the same context and are representative of the words teacher such as education, student, class, grade, testing or lessons.
“the further in education you go, the more work space the teacher has and less the student does.
That moment when a kid argues with teacher and entertains the whole class.
My mom got mad at my elementary school art teacher for teaching us how to draw Bart in class.
Dear Sophomore year of high school English teacher -no clue how you got my email address, but don’t send me political emails.
Certified high school Algebra 1 and Geometry teacher so. I can tutor as well
If anyone knows of a decent drum teacher let me know, he’d like some lessons.
I don’t want to go to school because the teacher is always angry and the principle is rude.
My third-grade teacher told me I would write novel someday.
In States high stakes testing and teacher evals have been going on for number of years.
A Government School teacher who is creating a library in every home.

Distributional Semantics:

Step 1: Summarize the occurrence statistics for each word in a large document.
In the distribution semantics pipeline the current from a large set of contexts are reduced to summary statistics row count for each word then these statistics can optionally be transformed.

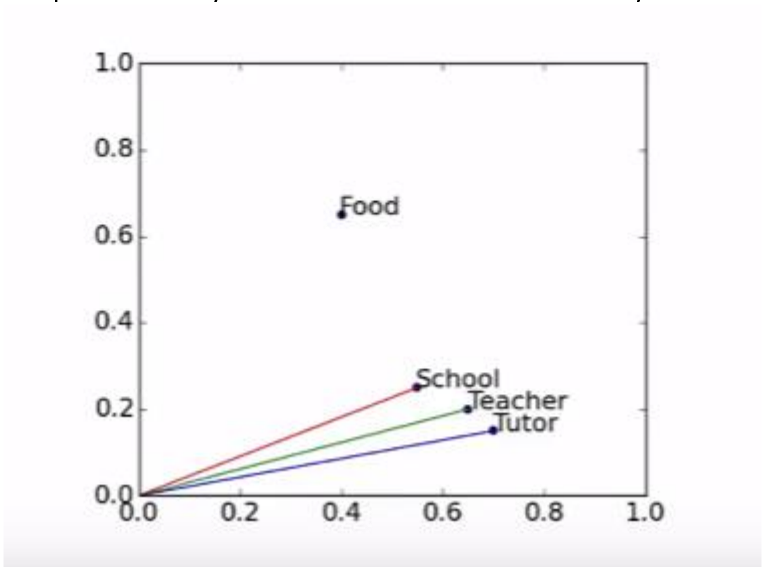
	Education	Student	Class	Lessons	Grade	Testing	School
Teacher	11	25	20	12	33	6	41

Step 2: Apply some transformation to the counts.
Here each word is represented by a vocabulary of length which is supposed to be sparse and containing redundant information using dimensionality reduction technique such as singular value decomposition or SVD results in a compact and dense real valued vector for each word.

	Dim 1	Dim 2
Teacher	0.65	0.2

e.g. dimensionality reduction (SVD) to obtain dense real-valued vectors.

Step 3: Compute similarity between words as vector similarity.



These vectors can be thought of represent each word in an n-dimensional Euclidean space. In this space words that are closer to each other are more similar. The figure above has 4 words embedded in a two-dimensional space. The word “Teacher” is closer to “Tutor” but also relatively closer to School. “Food” on the other hand is far away from all other words. The word similarity measure is usually used is the cosine similarity. Intuitively this represents the angle between the vectors joining the origin to the vector representations of the word.

Prediction based method:

In the prediction-based word the goal is to predict the word represent the SAP as numerical vector by using the vector of the words in its context. This prediction is repeated across all contexts in the dataset. In the end, similar words are predicted by similar contexts.

My mom was not happy at my	Elementary	School	Art	Teacher	For	Teaching	Us	How to draw …
My mom was not happy at my	Elementary	School	art	Teacher	For	Teaching	Us	How to draw..

All models offer a range of parameter that need to be specified.

- Underlying document set: The big set used for training the model plays a vital role Some datasets like medicine domains may offer different semantics than general purpose ones like Wikipedia.
- Context size: The Context size makes an important role in the similarity type obtained with larger contexts leading to more topical similarity with more restrictive complex leading to semantic and syntactic similarity.
- Context type: The context can be filtered to exclude frequent words or keep only certain parts speech or dependency types.
- Even others for count methods: counting method the type of matrix being built in first step. The method with which vector similarity is computed and be changed as well as the dimensionality reduction technique used.

Benefits of Tensor flow over other tools: (contributed by Hasitha Gajjala)

1. Graph Visualization: Tensor flow has very good computational graph visualizations when comparted to other tools like Torch and Theano.
2. Library Management: Tensor Flow was originally developed by Google. Because of its support from tech giant, it has very good performance, quick updates and new releases with new features.
3. Debugging: Tenor Flow allows users to execute the subparts of a graph where it gives an easy way to introduce and retrieve discrete data on an edge/subgraph and as such offers great debugging method.
4. Scalability: Tensor flow can run on wide range of hardware machines, from mobile devices to complex production environments.
5. Pipelining: Tensor flow supports highly parallel systems and designed to use various backends software like GPU, ASIC etc.

Limitations of Tensor Flow: (contributed by Manojit Saha Sardar)

1. Missing Symbolic Loops: Tensor flow does not support symbolic loops, the feature that is most required when it comes to variable length sequences. But, there is a workaround using finite unfolding.
2. No windows support: Tensor flow works well in a Linux environment but does not support windows users. Workaround is to use python package library, pip.
3. Computational Speed and Usage: Tensor flow lags behind in both speed and usage when compared to its competitors.
4. No GPU support: Tensor flow only supported GPUs are that of NVIDIA and the only language it supports is Python as there are tons of other languages in deep learning.