A project walkthrough based on Sentiment Classification (basic) to Text Summarization(advance).

Abstract: The work accomplished here in two different tasks. Sentiment classification for basic understanding of machine learning algorithms from probability and statistical view as well as deep learning terminology by addressing new improvements like Artificial Neural Network(ANN). Words can have many shades of emotional meaning. If we consider sentiment classification, they're simplified into three categories: neutral, positive, and negative. Beginning from basic probability based methods(Naive bayes) to advance architecture(DNNs) are used in this section.

Later, we introduce the Recurrent Neural Network(RNN), an advance architecture specially developed for sequential events like Text Summarization and why it's overperformed ANN? To incorporate with sequence, require advanced thoughts for the nature of it's distribution(time), some may be live and make computation complex. Now System/model architecture has evolved numerously, beneathing a reward of fast and facilitated computation. Transformer is one, capable of parallel computation while RNN is a class of artificial neural network where connections between nodes form a directed graph along a sequence. Otherside, Gensim, a nlp framework, offer's process of large datasets and streams, supports deep learning, and features unsupervised language modeling.

Transformational generation such as concise and fluent summary with the constraint of Meaningful key content information and keeping steady nature of overall meaning is worth summarization. Also, The goal of this system of art.

Keywords: Conditional Probability, Naive bayes, ANN, Seq2Seq, RNN, Word embedding, Bi-RNN, GRU, LSTM, Attention, Transformer, Trax, Matplotlib, NumPy, Tensorflow, Colab.

1.Introduction: Currently, we've an enormous amount of data from databases, warehouses and recently Data lake. So, it's necessary to classify texts generated from different perspectives. A class(positive/negative) can be identified by tedious ways, such as using their conditional probabilities. These categories can be numerically estimated with the help of conditional probabilities simply from frequency tables derived from large vocabulary. Probability theory is essential for tasks like Naive Bayes' or baseline for binary classification. Here, we'v added a notebook of Sentiment with Deep Neural Networks for basic natural language understanding and common preprocessing steps performed in the text data.
My first work here.

As you know, it relies on word frequency counts, we also can use a different form of word vectors that can give better results for our task.

Here, we'll inherit advanced development of RNN terminology for sequential tasks . In our case, here we'll mention the concepts needed for our tasks.

Now, most of the information is redundant, insignificant and may not convey the intended meaning. Thus, automatic text summarization allows people to get insights easily and rapidly. As well as it allows for more information to be fitted in a particular area. Though extractive summarization till now exists, now abstractive summarization and real-time summarization has become the great research area for it's complex and complicated nature. It is of two types: Extractive or Abstractive. Extractive reveal phrases or sentences, wherein abstractive only retrieve new sentences. It's necessary to address semantic analysis and lexical analysis for a better summary generation.

Primarily we'll discuss, to implement a uni-directional RNN in our encoding layer and attention in the decoder layer to generate some great summaries. Second work is here. It's possible to address reinforcement learning in it, as if it can learn from mistakes as well as improve the problem of data lackings.
Deployment of such models is easier now for technological advancement. We may deploy on the Web, it's now of much concern for mobile scanning devices. To automate the tasks of recognition(read) and summarization(writie) in a cross integrated model(image captioning, video activity recognition,e.t.c) version. Where CNN is used to find and make patterns available, then those patterns are used to sequence generation. Now, old fashion is backdated and in new terminology machines learned from huge amounts of text data and pairs. Seek to make new acceptable sequences from many structures. Probability based choice and vote strategy are applied.

2.Related Work: In this section, we'll establish the baseline used in models for abstractive summarization, then walk through modifications to the baseline that have achieved state-of-the-art results. Despite the clear differences between abstractive summarization and machine translation, the attentional RNN encoder-decoder model proposed in Bhadanau has become standard in abstractive summarization. For both tasks, the encoder generates a representation of the input, while the decoder uses these encodings to create the final outputs.

2.1 Baseline As a baseline model, Nallapati et al., (2016) proposes the use of a bidirectional GRU encoder, a unidirectional GRU decoder, an attention mechanism over the source's hidden states, and a softmax layer over the vocabulary to generate target words. Nallapati then adds a variety of features such as the large vocabulary trick, feature-rich encoding of keywords, and use of pointer networks to model rare words.

2.2 Large Vocabulary Trick Speeding up convergence and addressing the computational bottleneck caused by the softmax computation over the entire vocabulary, the "large vocabulary trick," proposed by Jean et al., (2014) restricts the decoder vocabulary of each mini-batch to words in the batch's source documents. It then adds words from the target dictionary until the

decoder's vocabulary reaches a set size. This approach focuses the model on words that come from the source, making it effective for summarization.

2.3 Out of Vocabulary Pointer Training the model requires restricting the decoder vocabulary, so it will inevitably run into words it does not recognize. Many models handle this by outputting an "UNK" token in place of the unknown word. Instead, Nallapati et al., (2016) allowed the decoder to choose, at each timestep, between producing a word in a regular fashion and generating a pointer to some source word, which is then outputted in the summary(rare word).

3. Literature review

3.1 Text representation

Many machine learning problems use text as an explanatory variable. Text must
Be transformed to a different representation(one_hot,bag_of_words, TF-IDF,word2vector hashing trick, Word_embedding) that encodes as much of its meaning
as possible in a feature vector. Several representations are reported which are:
Letting vocabulary have 10,000 words. Each word in a sentence is encoded with a 10,000-dimensional vector.

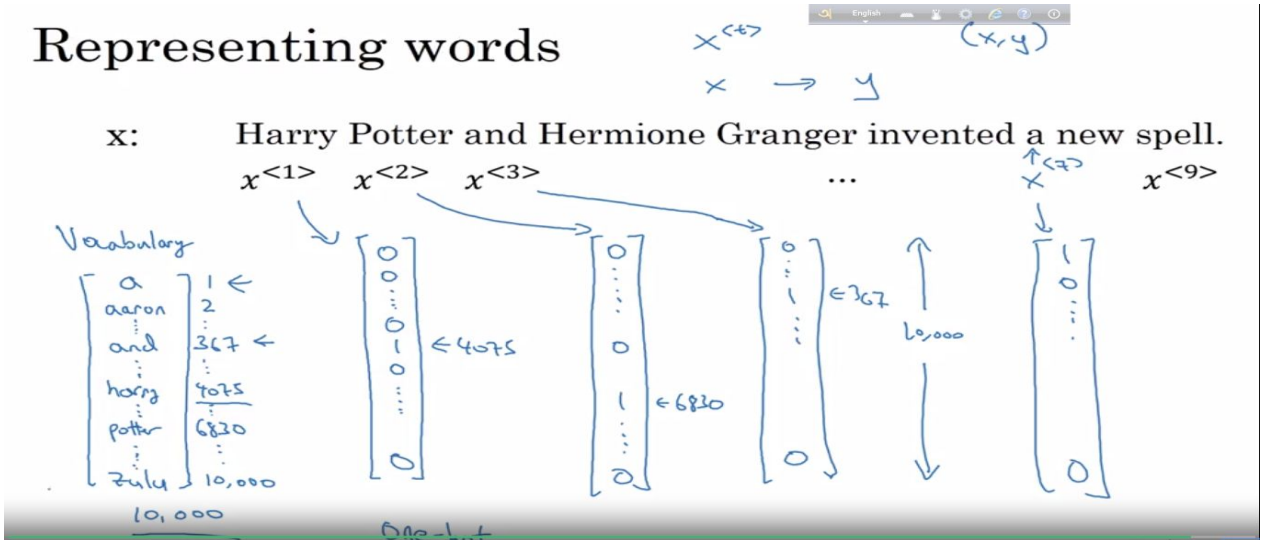


Fig: One-hot

Previously, Text was represented with One-of-k or One-hot encoding. With the extension of, naming the 'bag of words' model. Where 'bag of words' model is better for domain-specific(More informative than word_embedding)small dataset and 'word embedding' is something following where we wanna find the similarity among words and featurized it with lower dimensional vector than one-hot encoder.

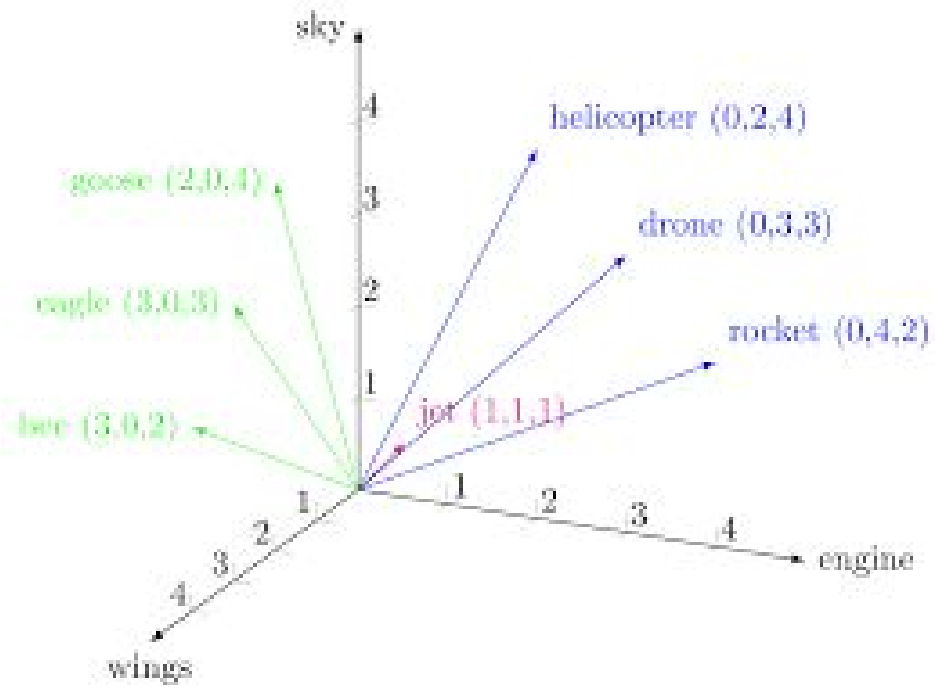


Fig: embedded word

Simply, similar Words appear in similar geometrical space. We can check it in low dimensional spaces(a method for this is PCA, t-SNE). This helps us build word embedding for many task like language_modeling, classification, machine_translation_model e.t.c.

For better intuition:

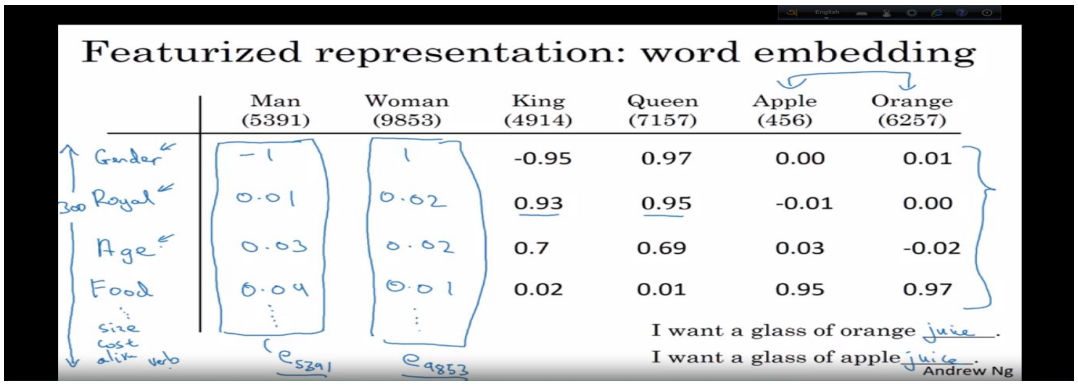


Fig-A: word embedding intuition

The size of embedding layer(voc_size, embedding_size) can be treated as the hyperparameter of the model, the embedding layer in the model is useful for good representation of the word and improvable.

Following are the common preprocessing steps performed in a text dataset.

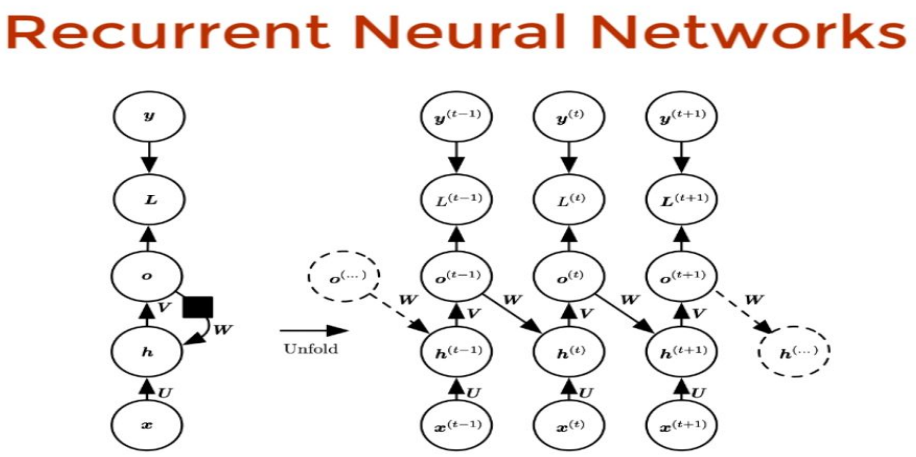
Stopword deletion, Stemming, lemmatization & Tokenization. Tokenization here reported for giving a solid way to first numeral representation of the context.

3.2 Tokenization: Tokenization is the process of splitting a string into tokens, or meaningful sequences of characters. Also possible for, two or more characters word.

Keras provides a Tokenizer class that can be fit on the training data a word to integer mapping w.r.t frequency. It can convert text to sequences consistently by calling the texts_to_sequences() method on the Tokenizer class, and provides access to the dictionary mapping of words to integers in a word_index attribute. Something like position defining in a meaningful sequence attribute.

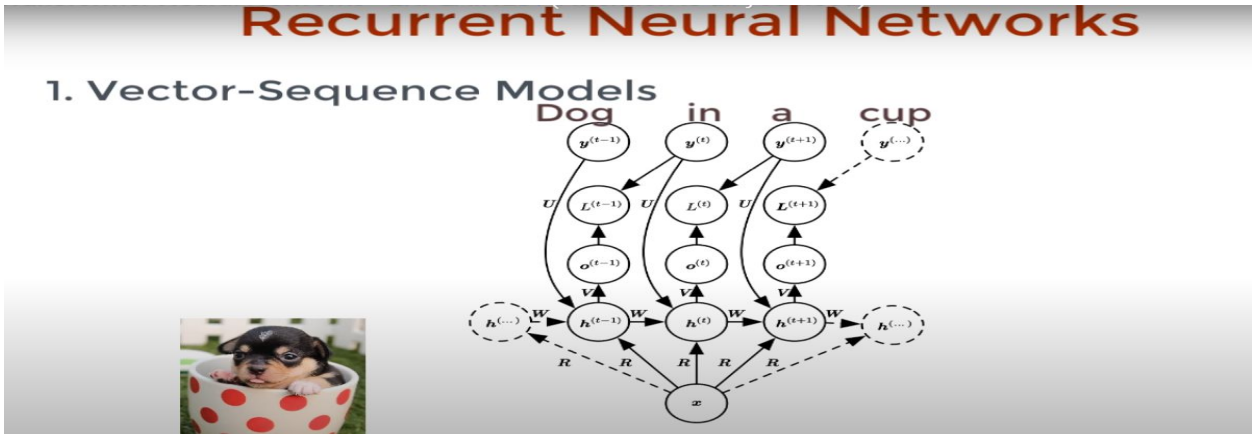
In the following section, we'll mention RNN.

3.3 Overview/Flashback:
This section is collected from here.
Recurrent neural nets, that are feed-forward neural networks that are rolled out over time.



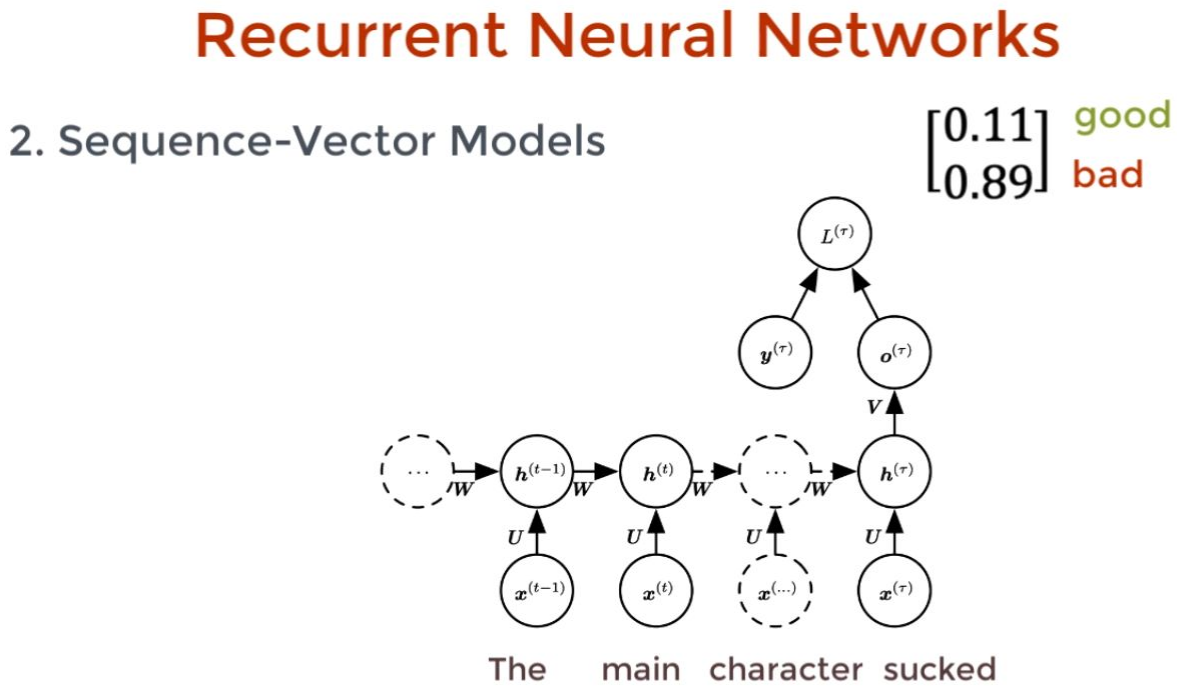
Here, x is input sequence, h is hidden states, t is the time stamp, W, V parameter.
As such they deal with sequence data which have some sort of ordering. This raises several types of architectures.

i. Vector-Sequence modeling:

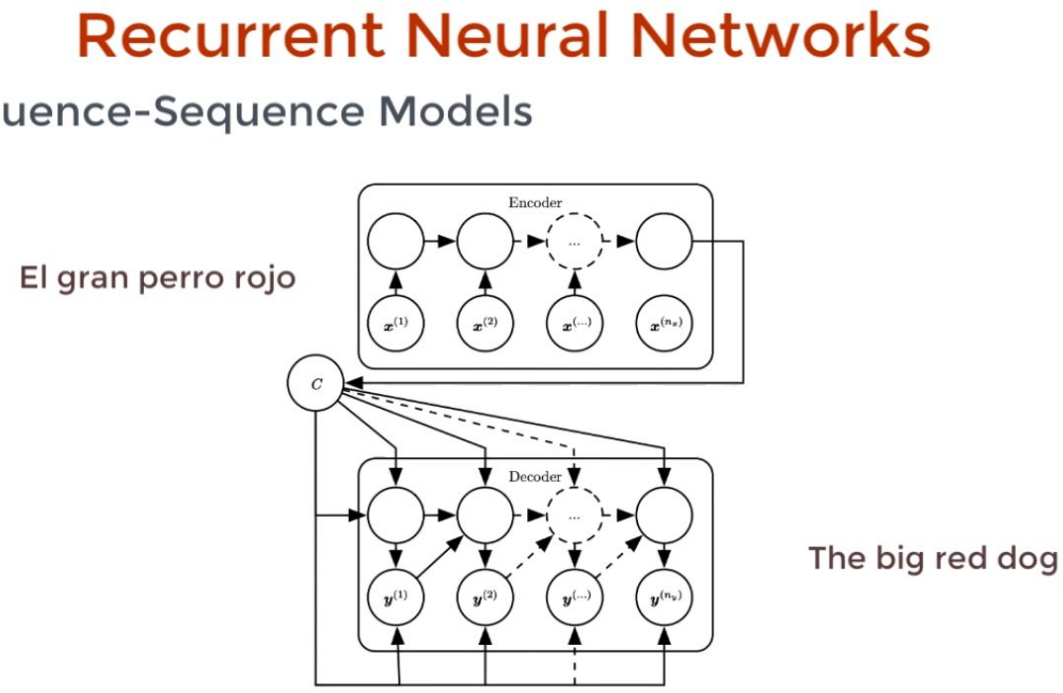


Input is the image features vector and output is the sequence of word(image captioning)

i. Seq to vector modeling



Here, input is a sequence of texts and output is a vector, something like movie review.



Language Translation is one kind of task.

The architecture mentioned above having different notations than below. As in the above we try to recognize with basic nlp applications.

All of the architectures following are collected from the internet.

3.4 Deep dive into the state of the art of available model

In, Seq2Seq modeling, basically RNN(Recurrent Neural Network) is used rather than ANN(Artificial Neural Network), For the following reasons:

Why RNN?
I. Inputs, Outputs can be different lengths in different examples, thus architecture needs to be varied concurly!
II. Doesn't share features learned across different positions of text.

3.4.1RNN:

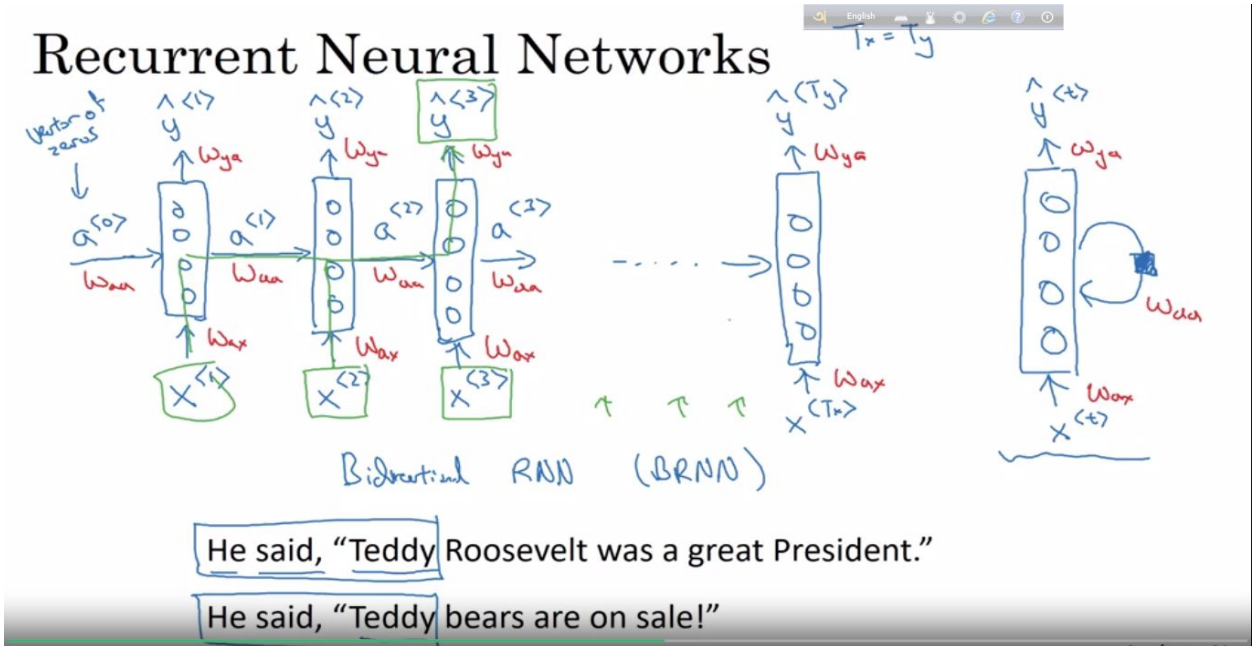


Fig: Unidirectional RNN in the forward pass

Here, the last column of the row is representing a unidirectional RNN architecture, each circle in it is a simple single RNN unit/layer. Where $x^{<t>}$ representing the input feature vector(character level/word level) representation and y^t representing the output at timestamp t, a is the simply computed activation in each time stamp for a layer, n_x vector length of feature input , W_{ax} Weight matrix multiplying the input, NumPy array of shape (n_a , n_x) , W_{aa} Weight matrix multiplying the hidden state, NumPy array of shape(n_a, n_a), W_{ay} Weight matrix relating the hidden state to the output, NumPy array of shape(n_a, n_y).

Three deep recurrent layers that are connected in time. You don't see as many deep recurrent layers as you would see in a number of layers in a deep conventional neural network like in the left one.

N-gram language model(give Prev N words) to predict the next word in a sequence, advance neural network(lstm) to generate text. In many applications you want to determine the sentiment of a sentence , With language modeling u solve problems infinitely. Translation, autocomplete, generating text from scratch, separating named entity recognition(people, place name). This is the building block of many nlp systems like identifying duplicate questions/answers in forums to set or sort them.

Basically, there are two phases in **language modeling**:

3.4.1.1 Training and Sampling

In the training phase, in fig(1) we're trying to predict the next word given the previous word(skip-gram)(understanding context-target pair) and other associated inputs.

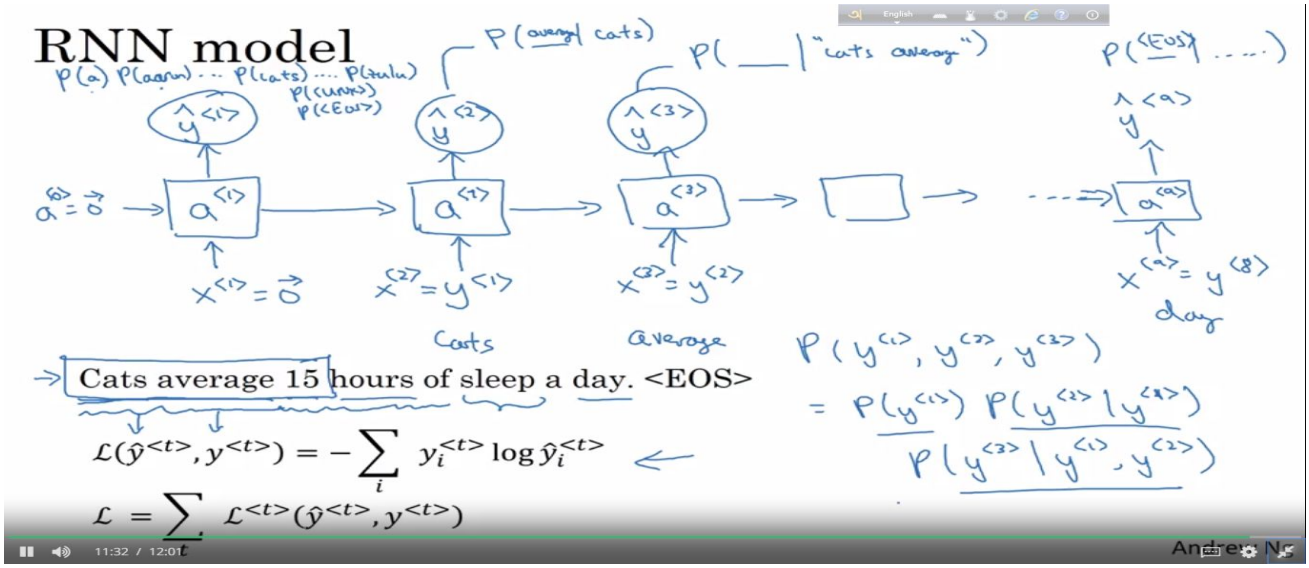


Fig: 01

The initial hidden state(a_0) can be assigned zero matrix or randomly, the next_state(a^t) can be obtained from the following equation in the picture below. Here, ReLU, tanh activation functions are used for calculation of hidden states and sigmoid used for prediction. Simply, learning context target pairs through forward and backward propagation in a loss minimizing intention with updating the parameters of the model.

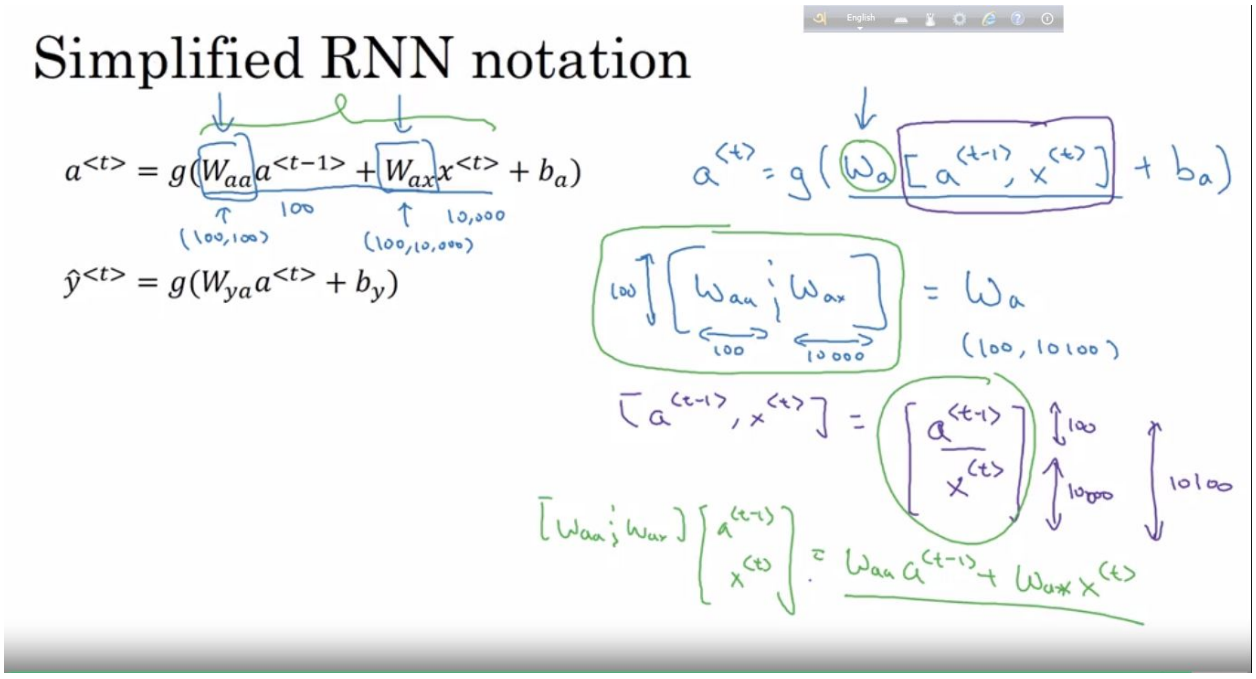


Fig: 02

On the right side, we simply vectorized the operation. In the training phase, we'r updating our parameters with the ambition of minimizing the cost function mentioned in Fig:01, using an optimization technique like gradient descent, adam optimizer, rmsprop e.t.c.

Better understanding for the computational process we recommend U, section intended for **multilayer network**.

In inference or sampling

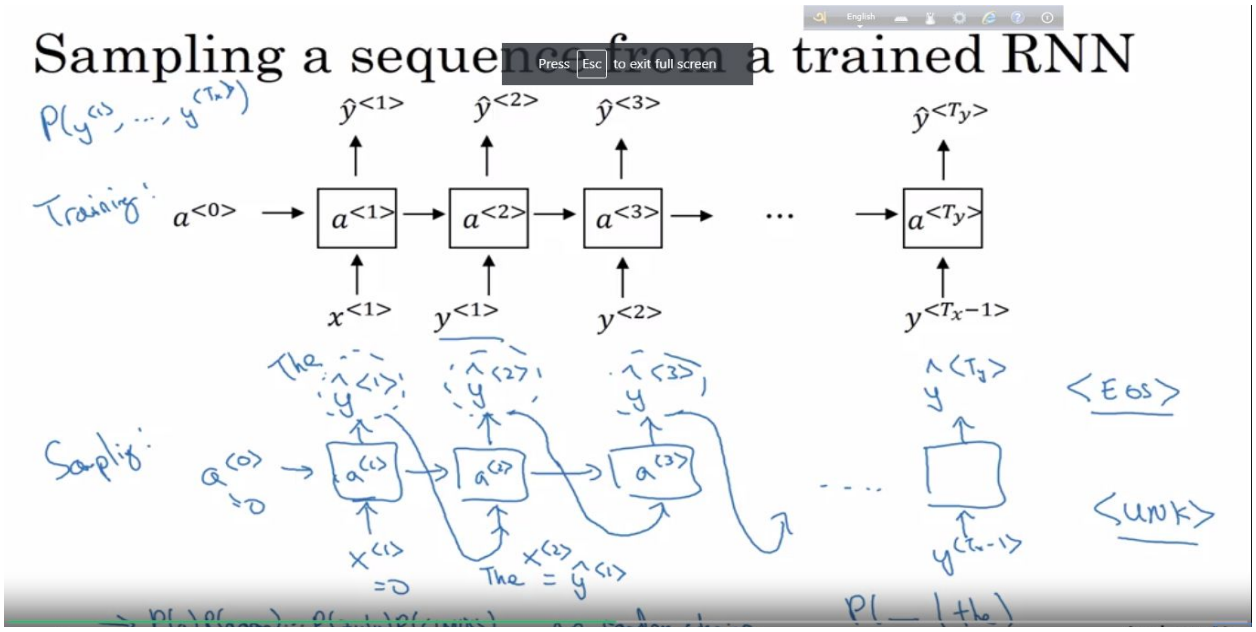


Fig:04

A softmax classifier is used on top of the architecture to choose the best word from vocabulary for natural language generation. Greedy search, beam search would be mentioned in this context.

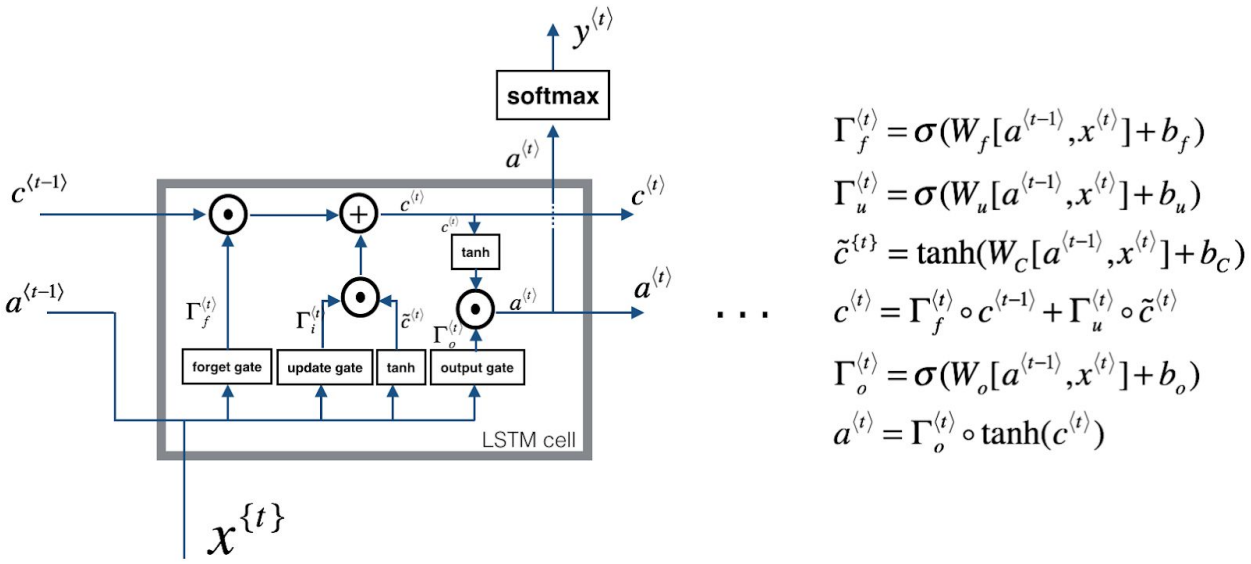
Cons:

I. Slower to train

II. Long sequence lead to vanishing/exploding gradients

Here context is simply the far previous words, also no prioritize/extra information(singular/plural e.t.c) is available for word from the previous or next context. But, this will work well enough for some applications, but it suffers from vanishing gradient problems. So it works best when each output can be estimated using mainly "local" context (meaning information from inputs where is not too far from. Thus LSTM(Long Short Term Memory) in action. which is better at addressing vanishing gradients. The LSTM will be better able to remember a piece of information and keep it saved for many timesteps.

3.4.2 LSTM: This cell has a branch that allows passed information to skip a lot of the processing of the current cell and move on to the next. This allows the memory to be retained for a longer sequence. Stacking LSTM on top creates the LSTM network. Similar to RNN but more complex.



$$\Gamma_f^{\langle t \rangle} = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$$
$$\Gamma_u^{\langle t \rangle} = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$$
$$\tilde{c}^{\{t\}} = \tanh(W_C[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_C)$$
$$c^{\langle t \rangle} = \Gamma_f^{\langle t \rangle} \circ c^{\langle t-1 \rangle} + \Gamma_u^{\langle t \rangle} \circ \tilde{c}^{\langle t \rangle}$$
$$\Gamma_o^{\langle t \rangle} = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$$
$$a^{\langle t \rangle} = \Gamma_o^{\langle t \rangle} \circ \tanh(c^{\langle t \rangle})$$

Fig-05: A single LSTM cell (can be repeatedly used)

It has functions of:

- Forget get W_f, are weights that govern the forget gate's behavior whether information will be carried or not.
- Update get will be used for updating the change in forget gate. Update the cell c^t with the help of candidate cell.
- Output gate is used to decide which outputs we will use.

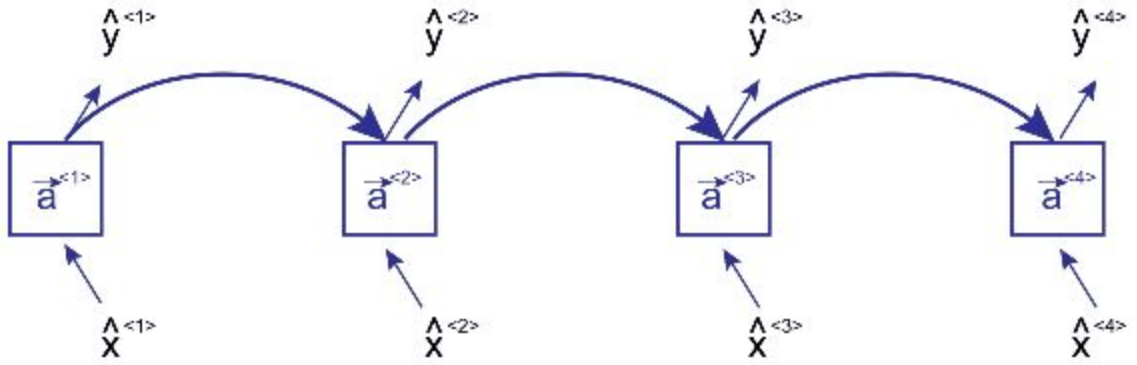
3.4.3 Bi-directional RNN network(Bi-RNN/LSTM/GRU):

This is a modification made on the normal RNN network to make it able to adjust to an important need in NLP problems, as in NLP , sometimes to understand a word we need not just to the previous word, but also to the coming word, like in this example

He said , "Teddy bears are on sale!"

Here to differ between the 2 different meanings of the word teddy (one time it is part of a person name, while the other is part of the word bear) we would need to look for the coming word, so this is the reason why we need to apply bidirectional networks. Bidirectional networks is a general architecture that can utilize any RNN model (normal RNN, GRU, LSTM).

This allows the prediction at each time stamp to take as input both information from the past, as well as information from the present which goes into both the forward and the backward things at this step, as well as information from the future.



Forward RNN (LSTM or GRU) network

Here we apply forward propagation 2 times, one for the forward cells and one for the backward cells. Both activations (forward, backward) would be considered to calculate the output y^ at time t.

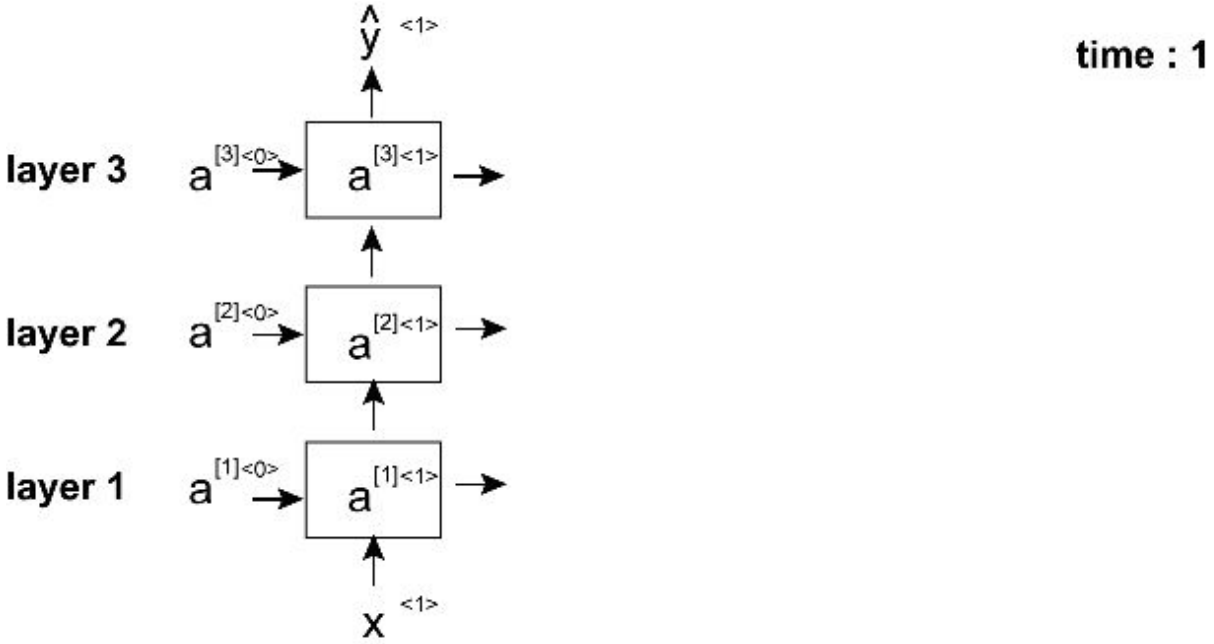
$$\hat{y}^{<t>} = g(W_y [\overrightarrow{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$$

Cons: The only disadvantage of the bidirectional RNN is that you do need the entire sequence of data before you can make predictions anywhere. Can be a problem for real time speech recognition tasks.

3.4.4 Multilayer network:

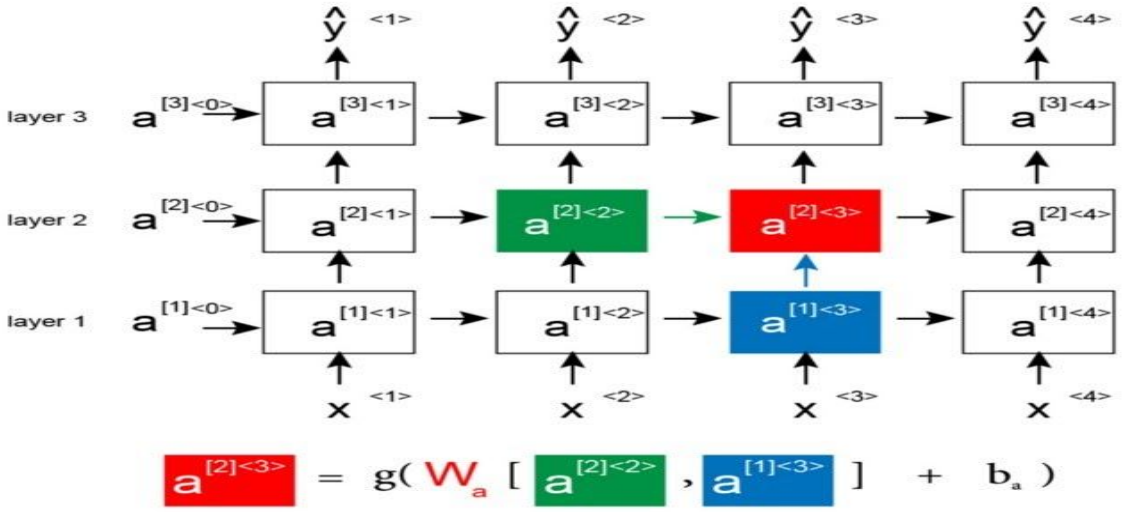
To achieve even greater results, we can stack multiple RNN(LSTM or GRU or normal RNN) on top of each other, but we must take into consideration that they work with time. As we can see, since we are working on RNN or its variations, we must take into consideration the time factor, so each vertical column of cells represents a layer, while each progress in time we repeat this column. So our notation would be

$$a^{[layer]\ <time>}$$



To get the value of any activation layer, we use both for the following picture:

1. Previous activation in time (time 2) from the same layer (layer 2) 💚 green
2. The previous cell at the same time (time 3) in the previous layer (layer 1) 🔵 blue



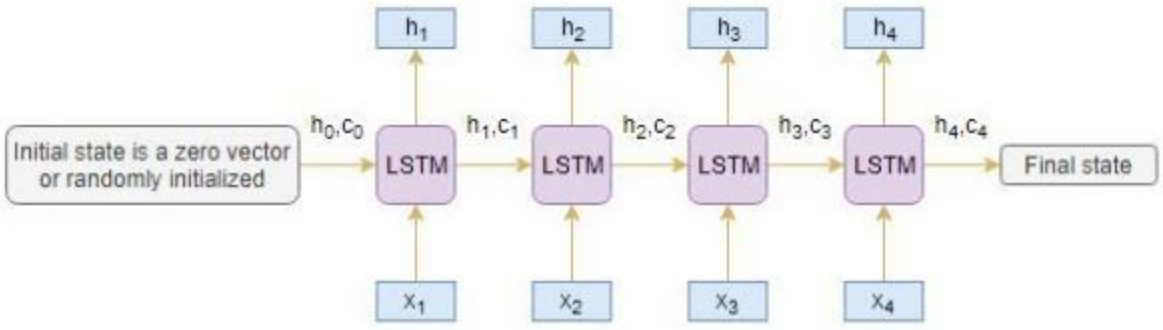
3.4.5 Two phases of the model for Text Summarization Task:

In the consideration of single-layer network:

i. Training:

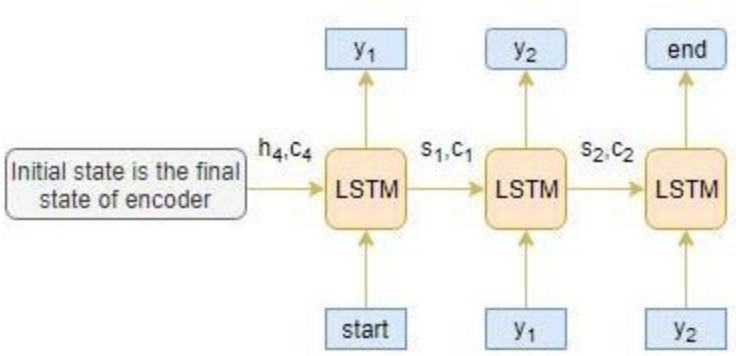
In the training phase, we will first set up the encoder and decoder. We will then train the model through forwarding and backward propagation of the network and try to predict the target sequence offset by one timestep.

Encoder: An Encoder Long Short Term Memory model (LSTM) reads the entire input sequence wherein, at each timestep, one word is fed into the encoder. It then processes the information at every timestep and captures the contextual information present in the input sequence.



The hidden state (hi) and cell state (ci) of the last time step are used to initialize the decoder. Remember, this is because the encoder and decoder are two different sets of the LSTM architecture.

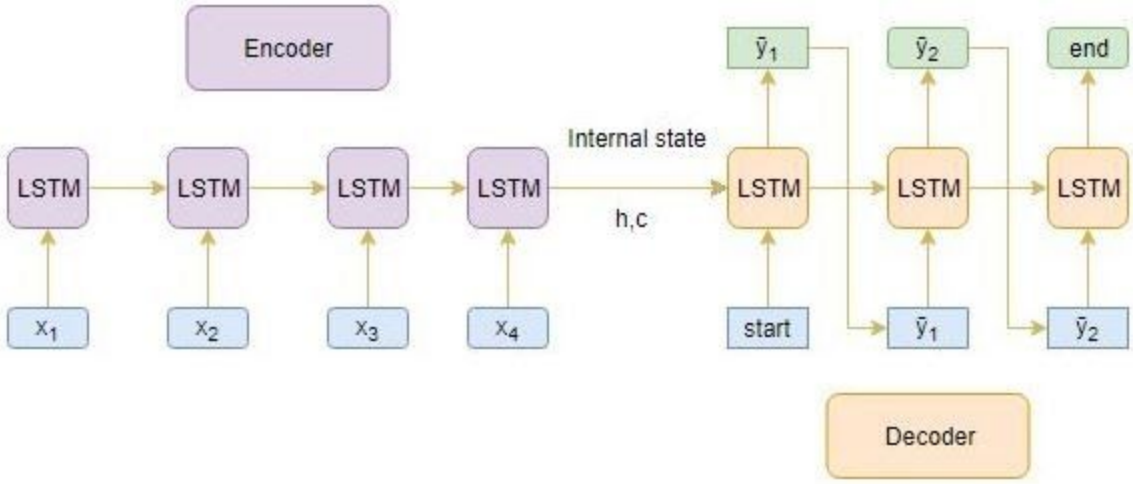
Decoder: The decoder is also an LSTM network that reads the entire target sequence word-by-word and predicts the same sequence offset by one timestep. The decoder is trained to predict the next word in the sequence given the previous word.



<start> and <end> are the special tokens which are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting the target sequence by passing the first word into the decoder which would be always the <start> token. And the <end> token signals the end of the sentence.

ii. Inference phase

After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence:



How does the inference process work?

Here are the steps to decode the test sequence:

1. Encode the entire input sequence and initialize the decoder with internal states of the encoder
2. Pass <start> token as an input to the decoder
3. Run the decoder for one timestep with the internal states

4. The output will be the probability for the next word. The word with the maximum probability will be selected
5. Pass the sampled word as an input to the decoder in the next time step and update the internal states with the current time step
6. Repeat steps 3 – 5 until we generate <end> token or hit the maximum length of the target sequence

Cons: “A potential issue with this encoder-decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences. The performance of a basic encoder-decoder deteriorates rapidly as the length of an input sentence increases.”

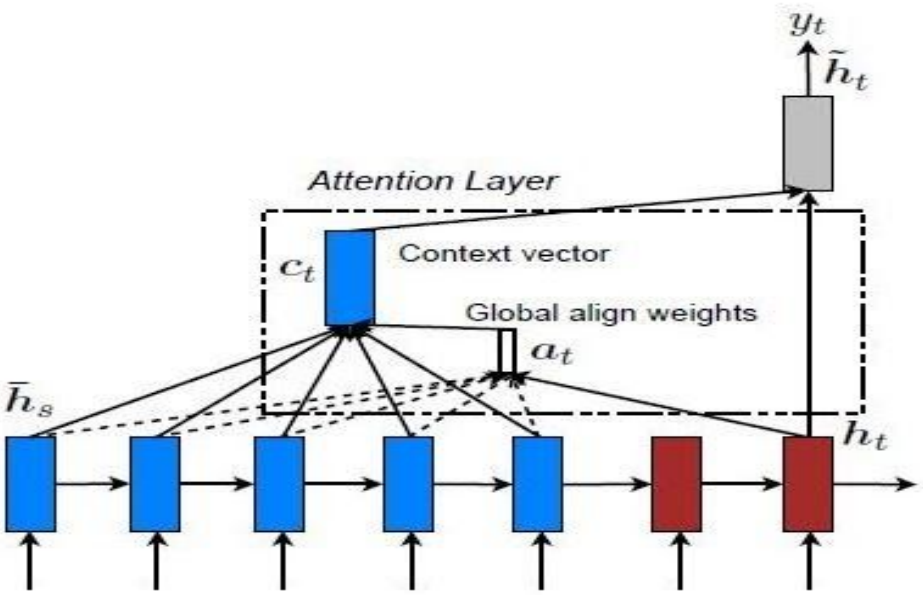
-Neural Machine Translation by Jointly Learning to Align and Translate

3.4.6 Attention:

The intuition behind the attention mechanism: How much attention do we need to pay to every word in the input sequence for generating a word at timestep t? That’s the key intuition behind this attention mechanism concept. Basically , two types of attention depending on the way the context vector is derived.

For encoder-decoder neural networks, the use of attention allows for the creation of a context vector at each timestep, given the decoder’s current hidden state and a subset of the encoder’s hidden states. For global attention, the context vector is conditioned on all of the encoder’s hidden states, whereas local attention uses a strict subset of the encoder’s hidden states

Global attention: Here, the attention is placed on all the source positions. In other words, all the hidden states of the encoder are considered for deriving the attended context vector.



Local attention: Here, the attention is placed on only a few source positions. Only a few hidden states of the encoder are considered for deriving the attended context vector:
However, LSTM is even slower they are more complex. Cause, all of the above architecture needs to be passed sequentially or serially one after the other. We need input from the previous state to make any operations on the current state.
Such sequential flow does not make the use of today's GPUs very well which are designed for parallel computation. Using parallelization for sequential data a Transformer neural network architecture was introduced.

3.4.7 Transformer NN architecture:

Transformer Components

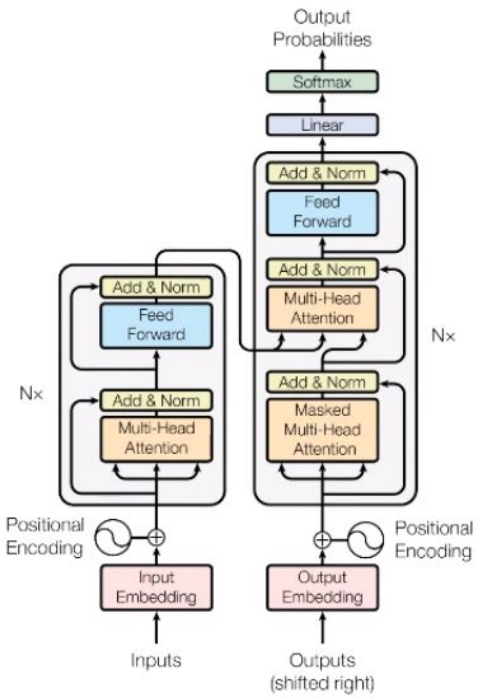


Figure 1: The Transformer - model architecture.

Transformer neural nets replaced all of the tasks mentioned in the Flashback section. For example google created BERT, which uses transformer to pre-train model for common nlp tasks. However, there is also a paper naming Pervasive Attention that could give a better results than transformer model architecture.

4. Preparing the dataset & model building:

Approach: We'll use RNN in our encoding layer and attention in the decoding layer. For attention, we'll use Bhadanau.

The sections of this main project are:

- Inspecting the data
- Preparing the data
- Building the model
- Training the model
- Making our own summaries

4.1 Inspection & Preprocessing of Dataset: we will be using fine food reviews from Amazon to build a model that can summarize text. Specifically, we will be using the description of a review as our input data, and the title of a review as our target data.

4.1.2 Text cleaning

1. Inspect the dataset in Colab
2. Lowercase conversion.
3. Drop nan and duplicate
4. Replace contractions with their longer forms
5. Remove any unwanted characters
6. Stop words will only be removed from the description text not from the summaries.

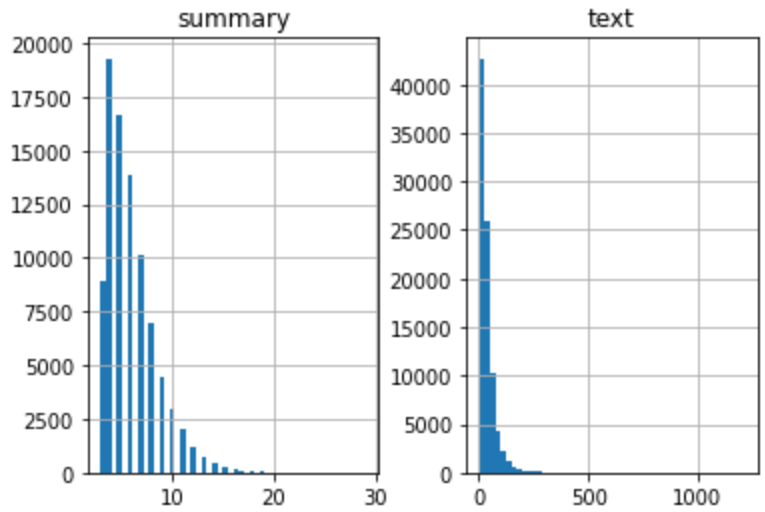
After cleaning:
Here are some review-summary pairs we've got:

```
Review: bought several vitality canned dog food products found good
quality product looks like stew processed meat smells better labrador
finicky appreciates product better
Summary: _START_ good quality dog food  _END_

Review: product arrived labeled jumbo salted peanuts peanuts actually
small sized unsalted sure error vendor intended represent product jumbo
Summary: _START_ not as advertised  _END_

Review: confection around centuries light pillowy citrus gelatin nuts case
filberts cut tiny squares liberally coated powdered sugar tiny mouthful
heaven chewy flavorful highly recommend yummy treat familiar story lewis
lion witch wardrobe treat seduces edmund selling brother sisters witch
Summary: _START_ delight says it all  _END_

Review: looking secret ingredient robitussin believe found got addition
root beer extract ordered made cherry soda flavor medicinal
Summary: _START_ cough medicine  _END_

Review: great taffy great price wide assortment yummy taffy delivery quick
taffy lover deal
Summary: _START_ great taffy  _END_
```

Here, <start> and <end> are the special tokens which are added to the target sequence before feeding it into the decoder

4.1.3 The next step is **understanding the distribution** of the dataset. In our case, understanding of the text length of reviews and summary will help us realize the overall distribution, thus we can select the maximum length of our input and output data sequences. It

will help us to reduce the padding & computational cost as it is many to many sequence modeling.



Interesting. We can fix the maximum length of the reviews to 80 since that seems to be the majority review length. Similarly, we can set the maximum summary length to 10.
Here, we can do EDA, Statistical probabilistic test e.t.c
Now, split the dataset into train/dev/test part. Here, we might face bias/variance problems as well as we can cope with it by adding more data, regularization, batch normalization, data collection and sampling in accordance with several statistical tests, like t-test, chi-square e.t.c.

For tokenization:
A method used fit_on_texts which Updates internal vocabulary based on a list of texts. This method **creates the vocabulary index** based on **word frequency.** So if you give it something like, "The cat sat on the mat." It will create a dictionary s.t. word_index["the"] = 1; word_index["cat"] = 2 it is word -> index dictionary so every word gets a unique integer value. 0 is reserved for padding. So lower integer means more frequent word (often the first few are stop words because they appear a lot).
texts_to_sequences Transforms each text in texts to a sequence of integers. So it basically takes each word in the text and replaces it with its corresponding integer value from the word_index dictionary. Nothing more, nothing less, certainly no magic involved. Why don't combine them? Because you almost always fit once and convert to sequences many times. You will fit on your training corpus once and use that exact same word_index dictionary at train / eval / testing / prediction time to convert actual text into sequences to feed them to the network. So it makes sense to keep those methods separate.

4.2 Building the Model:
We are finally at the model building part. But before we do that, we need to familiarize ourselves with a few terms which are required prior to building the model.

As, we'll use encoder-decoder LSTM(3 layers) in the encoding network and attention in the decoding network. The input is simply a 2D matrix of shape(number_of_training_examples, max_length_of_T.E), in embedding layer a third dimension is added for each word vector, in our case it's 500. Batch size is the number of training examples picked randomly and use to train our model, in our case it's 256.
In our case, we'll use rmsprop optimizer to update our parameter and sparse_categorical_cross_entropy for loss calculation. To prevent overfitting before convergence, we'll use early stopping to stop training.

4.2.1 Training:

Loss: When doing multi-class classification, categorical cross entropy loss is used a lot. It compares the predicted label and true label and calculates the loss. In Keras with TensorFlow backend support Categorical Cross-entropy, and a variant of it: Sparse Categorical Cross-entropy.

Sparse Categorical Cross Entropy:

Following is the definition of cross-entropy when the number of classes is larger than 2. As, our output class can be of equal vocabulary size for each time stamp. Sparse Categorical Cross-entropy and multi-hot categorical cross-entropy use the same equation and should have the same output. The difference is both variants cover a subset of use cases and the implementation can be different to speed up the calculation.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

This saves memory when the label is sparse (the number of classes is very large). Now how can we improve the speed? When labels are mutually exclusive, in normal cross-entropy calculation, there is a lot of log/dot/sum operations applied on y_pred probabilities that will eventually be 0s. We actually don't need those operations.

Optimization Alg:

Gradient Descent: **Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum of a function using gradient descent, we take steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. The gradient of differential function are calculated from backpropagation.**

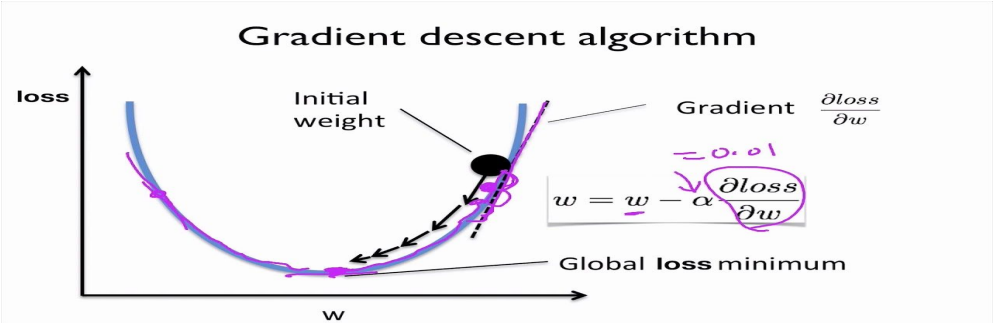


Fig: Loss function in 2d space

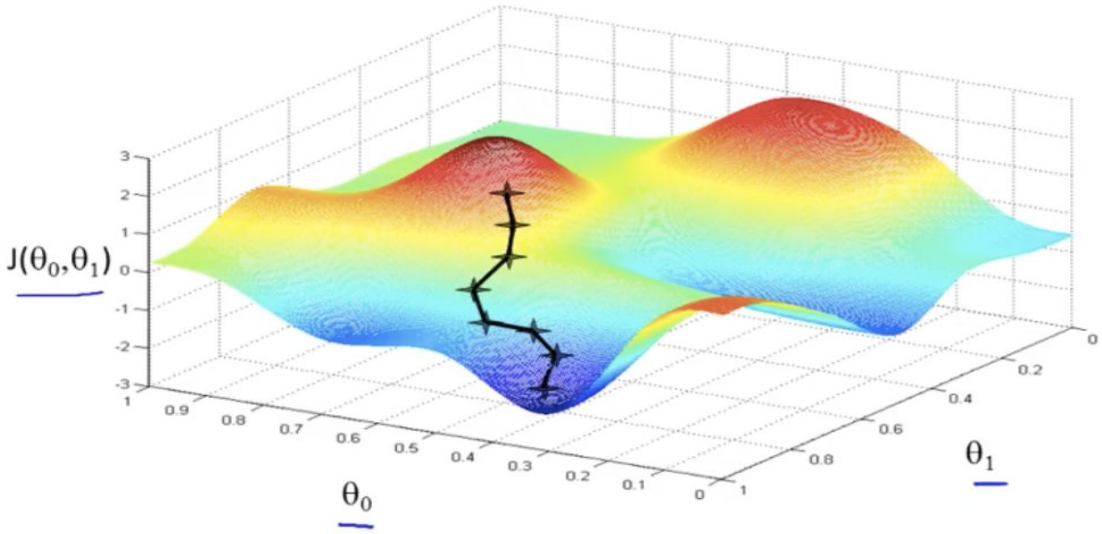


Fig: Loss function in multidimensional space

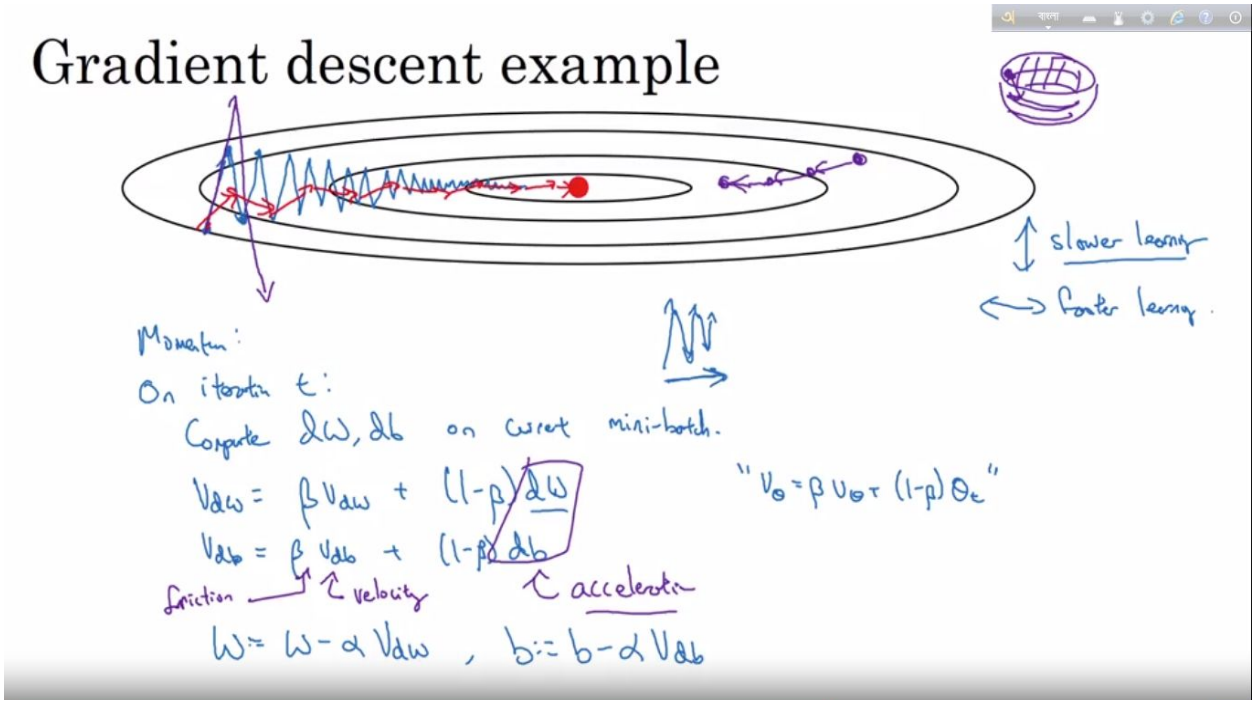


Fig: Gradient descent with contour loss space

In iteration t, it computes gradient on current minibatch and compute an exponentially weighted average of parameter. This allow us train smarter in less time and prevent from Jiggling in loss space and stucking into a local minima. Here, beta is a hyperparameter.

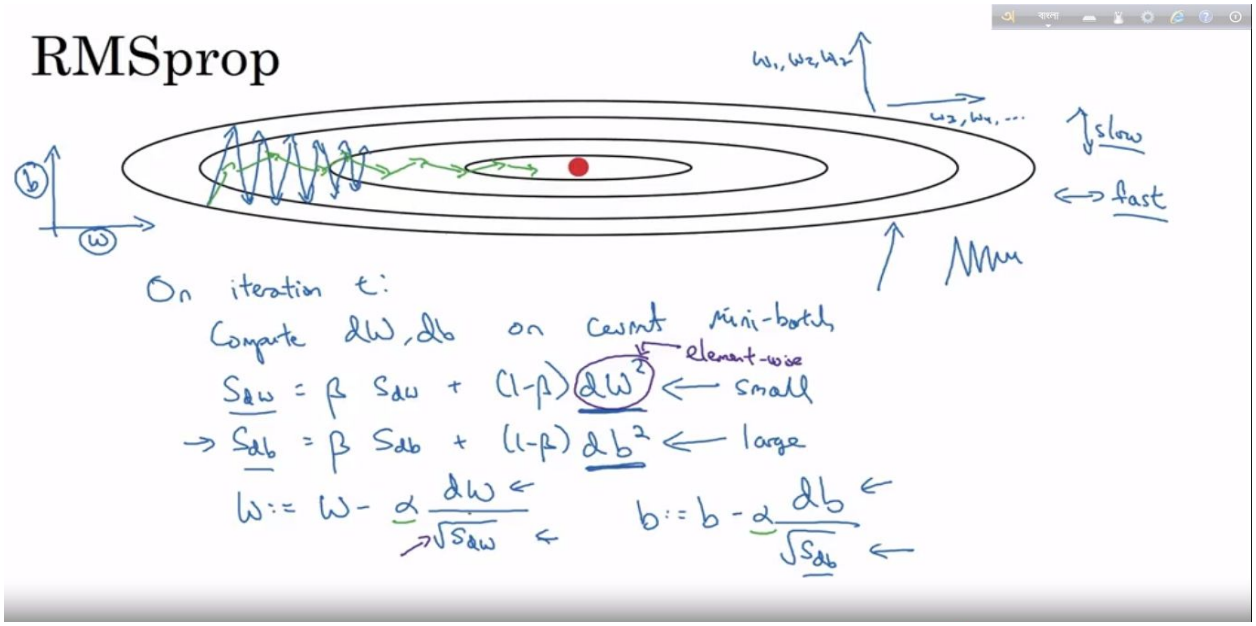


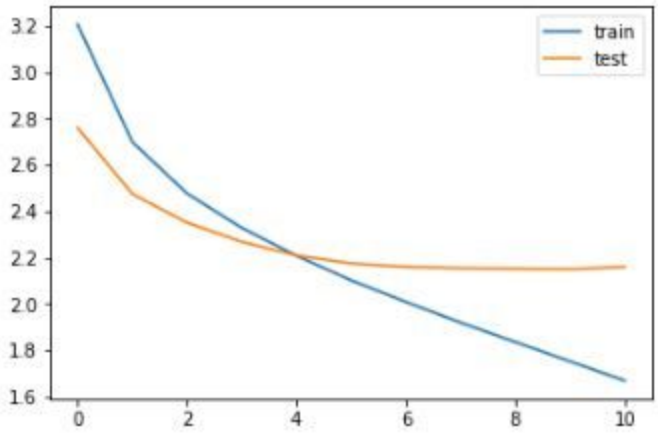
Fig: RMSprop optimization process

Here, the main difference is clearly shown above.

In deep learning:

```
If validation loss >> training loss you can call it overfitting.
Use more data, drop_out e.t.c

If validation loss > training loss you can call it some overfitting.
Use early stopping

If validation loss < training loss you can call it some underfitting.
Use a complex architecture

If validation loss << training loss you can call it underfitting
```

Early_stopping: In machine learning, early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration.



We can infer that there is a slight increase in the validation loss after epoch 10. So, we will stop training the model after this epoch.

The model already stopped training at epoch 6 and regarding

```
loss: 1.6066 - val_loss: 2.1301
```

As, we see val loss is higher than training loss, thus we can apply the following:
i. More training data
ii. A modern approach to reducing generalization error is to use a larger model that may be required to use regularization during training that keeps the weights of the model small.

a. Using batch normalization make train faster

iii. Use large validation data

iv. Addressing low bias and variance problems in the data may be tried k-fold, cross-validate to split the data into train/dev/test set.

These techniques not only reduce overfitting, but they can also lead to faster optimization of the model and better overall performance.

4.2.2 Inference:

For this section, we refer to the section **3.4.5 ii**

```
Review: organic usually prefer whatever blech cannot stand taste ended giving away going try another bag mention calories either be
ars calories take haribo please
Original summary: taste terrible
Predicted summary:  not that great
```

```
Review: package six boxes forty eight bags per box listed area large tea bags suitable making gallon time tea fact small single use
bags box web page says family size bags nothing family sized single use bags bad advertisement buy read misleading ads carefully ho
pe company business
Original summary: misleading advertisement
Predicted summary:  not as advertised
```

```
Review: mallomars pure chocolate cookies delicious tasty chocolate inside equally tasty cream filling inside pour ice cold glass mi
lk sit back try eat whole box one sitting brian fairbanks
Original summary: delicious
Predicted summary:  best chocolate have ever tasted
```

4.3 Summary:

We've implemented a sentiment classification project, a baseline of NLP. Refreshing from a very start basic vocabulary_building, mapping, numerical text representation, and applying naive bayes relies on word frequency count. Then we build an ANN architecture from scratch and apply a simple sigmoid for binary classification tasks. A softmax can be applied for multiclass tasks.

In advance NLP task, like text summarization we've built model1, simply we begin with unidirectional encoder-decoder LSTM with attention in decoder and randomly initialized word embeddings(Semantic vectors let you compare word meanings numerically) intuition from fig:-A. This word embedding can use the different vector similarity function(euclidian_distance, cosine similarity) to update the embedding matrix. However, after training we inferenced some great summaries by decoding test sequences in greedy approach(argmax) from this model.

For the model1, we hoped that the learned word vector representations would become more suited for summarization than pre-trained Word2Vec or GloVe vectors. However, it would take a high number of iterations on a large dataset to develop accurate word embeddings for the task.

Future Work:

How can we Improve the Model's Performance Even Further?

your learning doesn't stop here! There's a lot more you can do to play around and experiment with the model:

- We recommend you to **increase the training dataset size**, we can use the idea of **data generator, a** task described in the sentiment classification section, and build the model on that data. The generalization capability of a deep learning model enhances with an increase in the training dataset size
- Try **implementing Bi-Directional LSTM** which is capable of capturing the context from both the directions and results in a better context vector
- Use the **beam search strategy** for decoding the test sequence instead of using the greedy approach (argmax)
- **Evaluate** the performance of your model based on the **BLEU score**
- **Implement pointer-generator networks** and coverage mechanisms

Conclusion: Here, we've reviewed the different terminology associated with NLP. From word meaning to numerical mapping and how to perform Sentiment Classification tasks, addressing RNN/LSTM/GRU and starting from very basic sequence generation like language modeling(n-gram) to an advanced encoder-decoder architecture for sequence(text) summarization application. We don't stop here, we're working on it to make the model faster, cross-integrated, robust, reliable and may deploy on mobile devices to automatically scan, recognize and generate sum fluent summaries in real time!