# Sentiment Analysis: Using Recurrent Neural Networks

By Bibek Dahal

# Objective

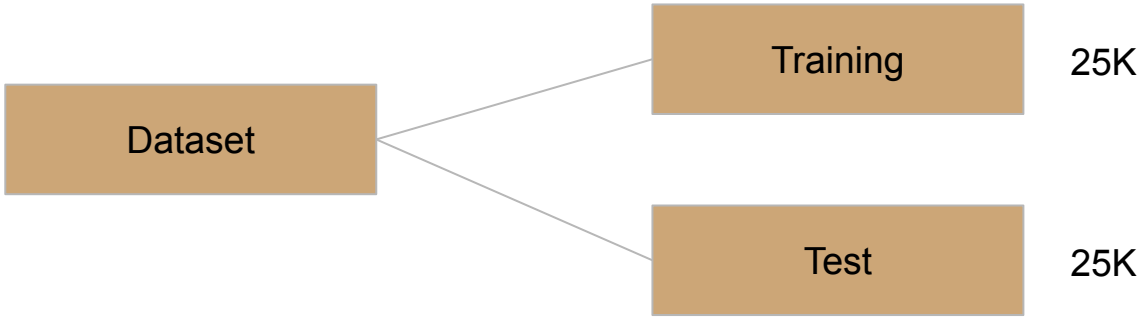
Text classification


**Analyze emotion of text's author**

# Dataset

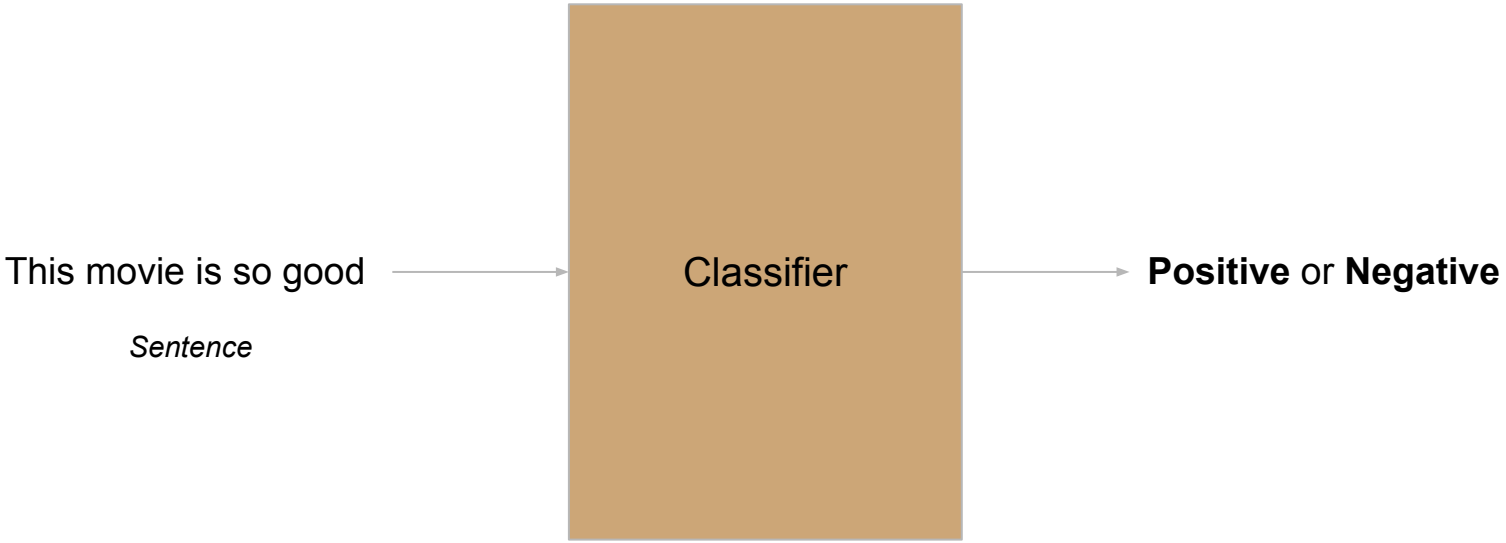
Stanford's Movies Review Dataset

https://ai.stanford.edu/~amaas/data/sentiment/

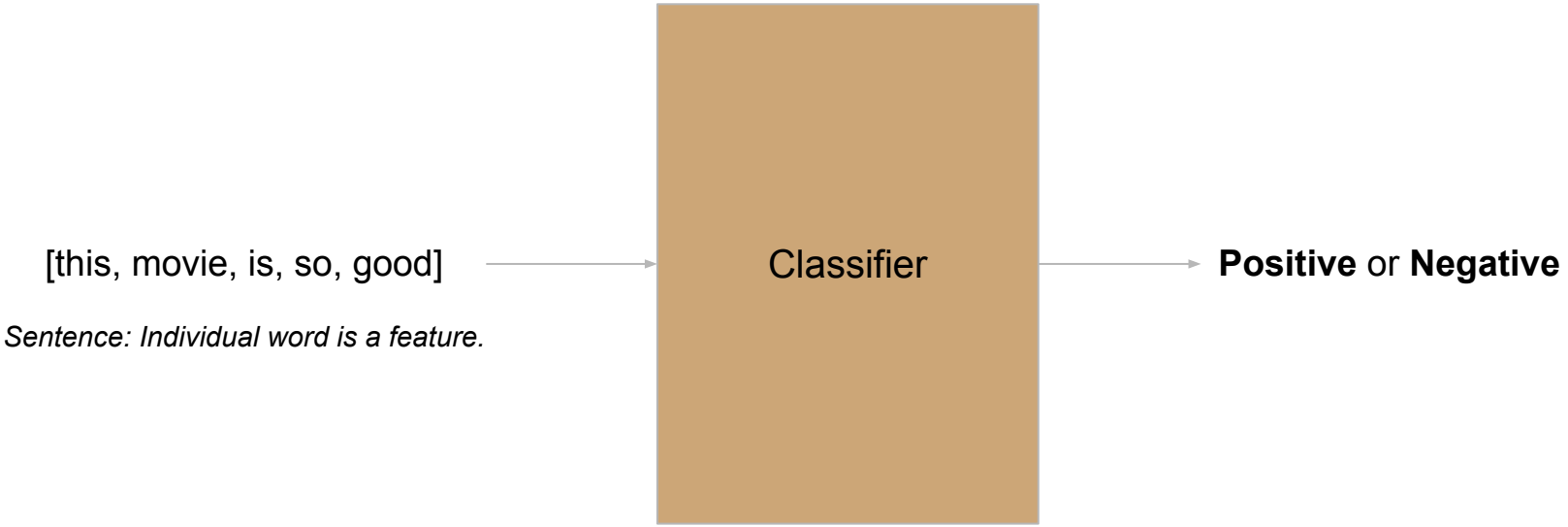



Labels: **Positive** or **Negative**
Equal number of samples for both.

# Approach


This movie is so good
Sentence
Classifier
Positive or Negative

# Approach



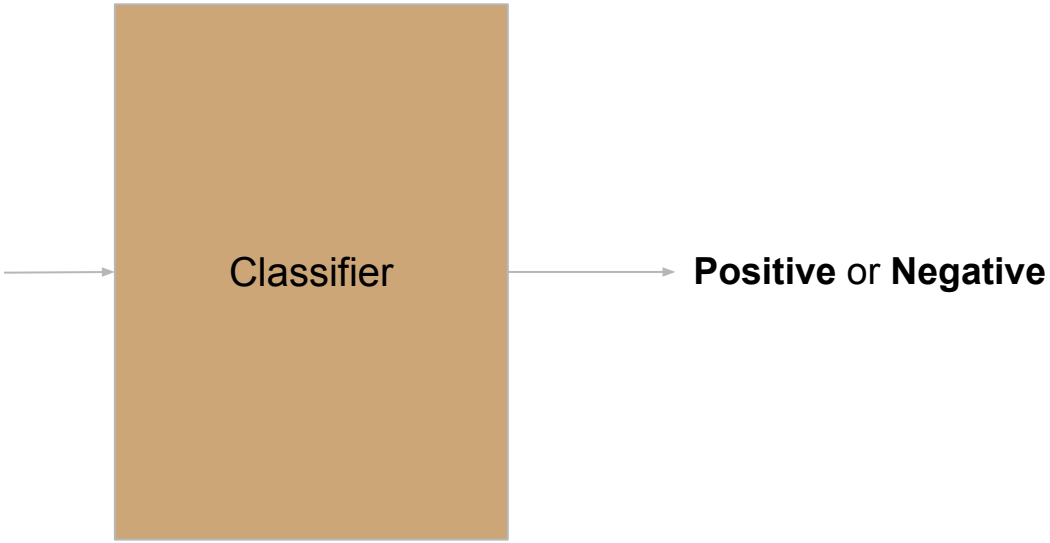

*Sentence: Individual word is a feature.*

# Approach

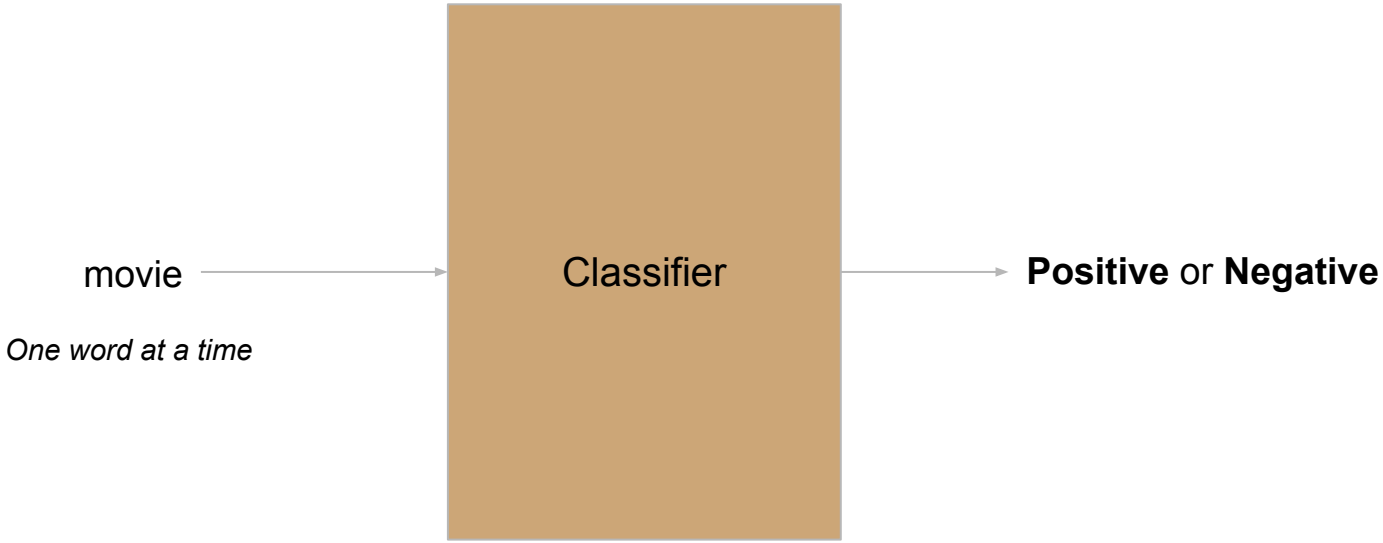
[this, movie, is, so, good]

[i, think, that, the, story, is, pointless, and, the, acting, is, bad]

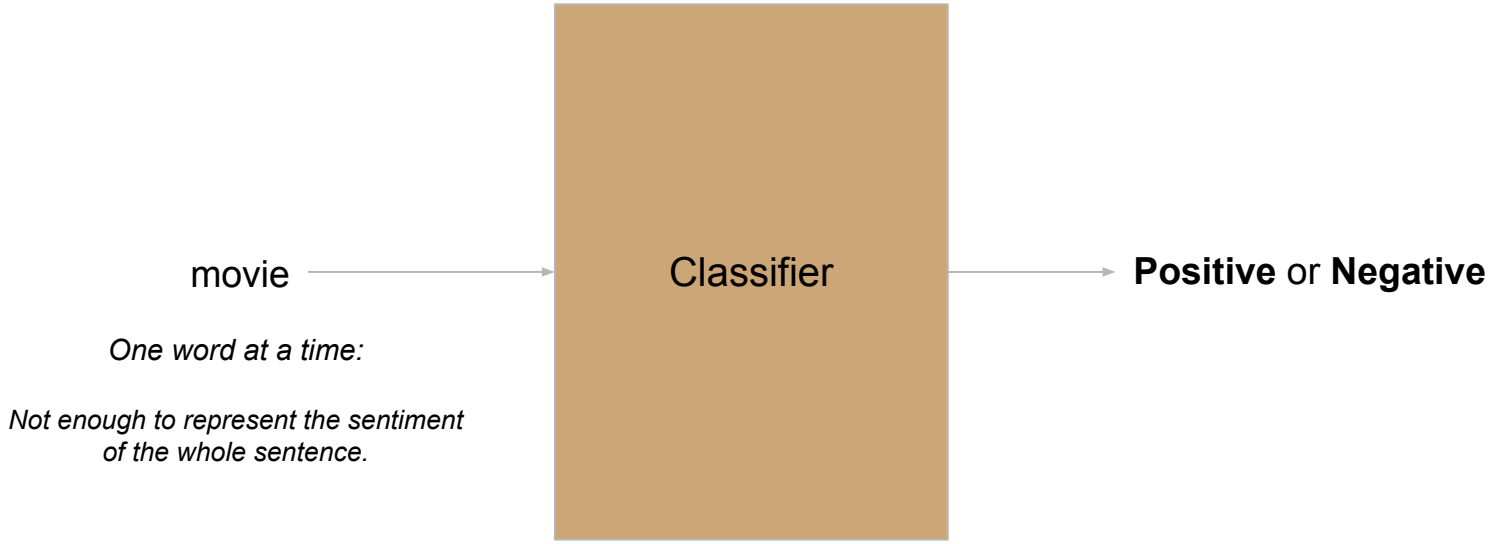
***Sentence:***
*Indefinite length*
*Important features at different position*



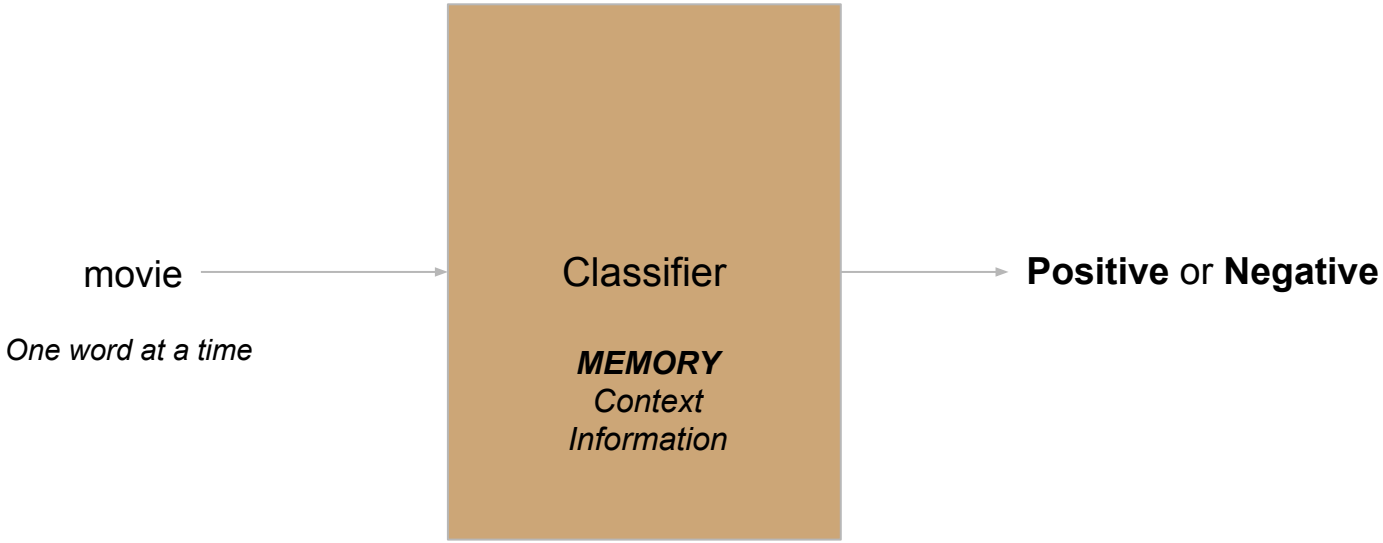
# Approach



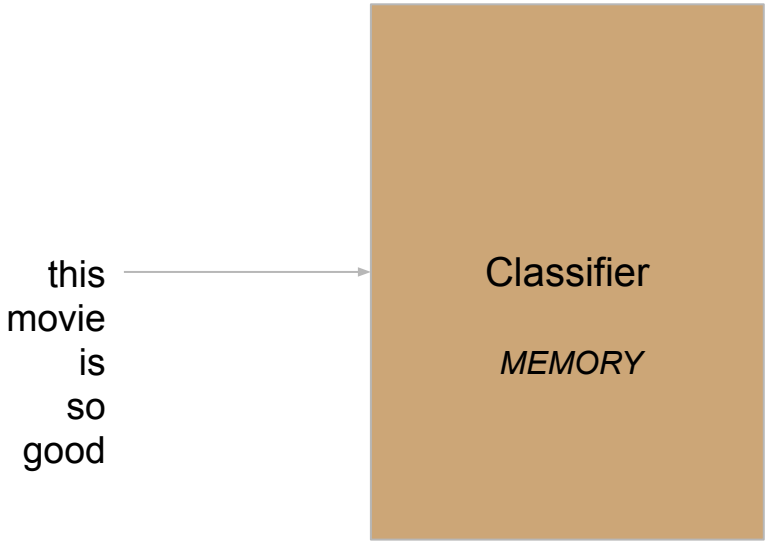
# Approach


movie
Classifier
Positive or Negative
One word at a time:
Not enough to represent the sentiment of the whole sentence.

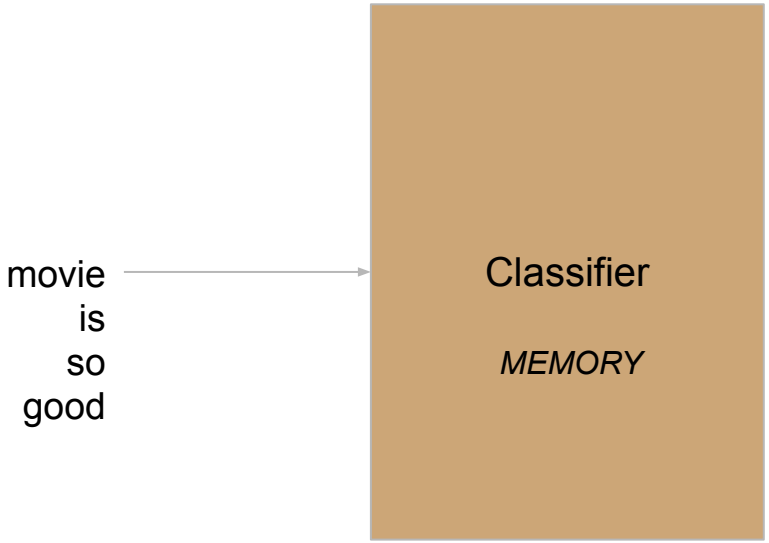
# Approach


movie
One word at a time
Classifier
MEMORY
Context
Information
Positive or Negative

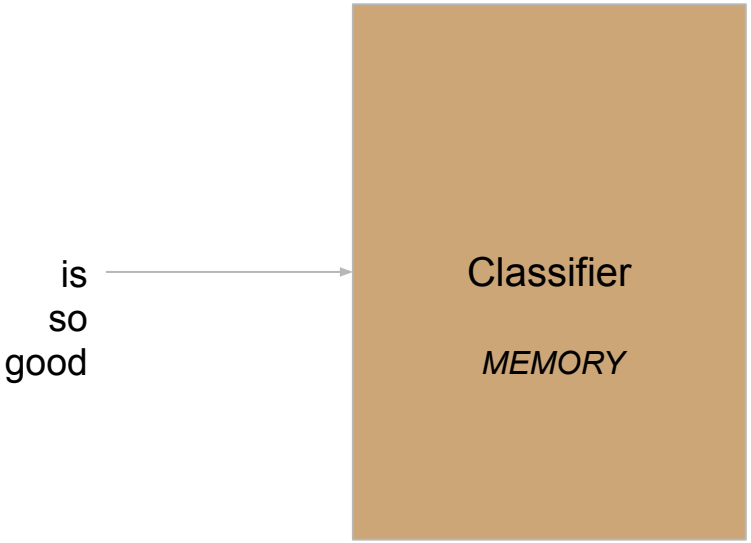
# Approach



# Approach

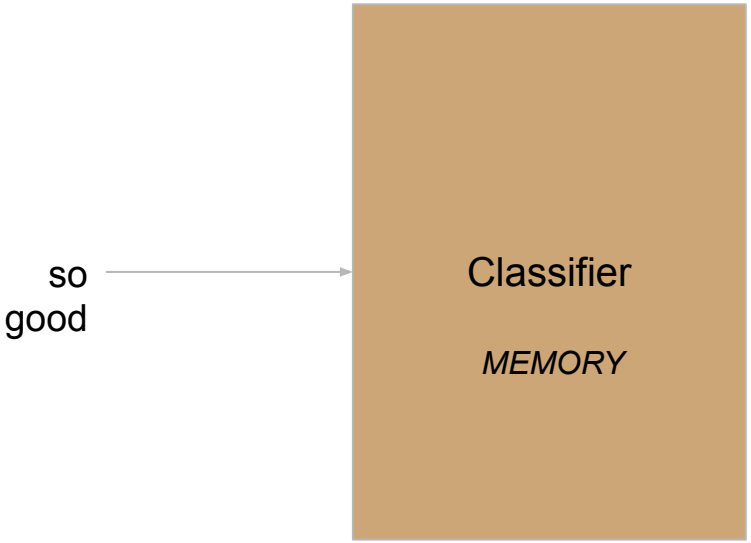


# Approach



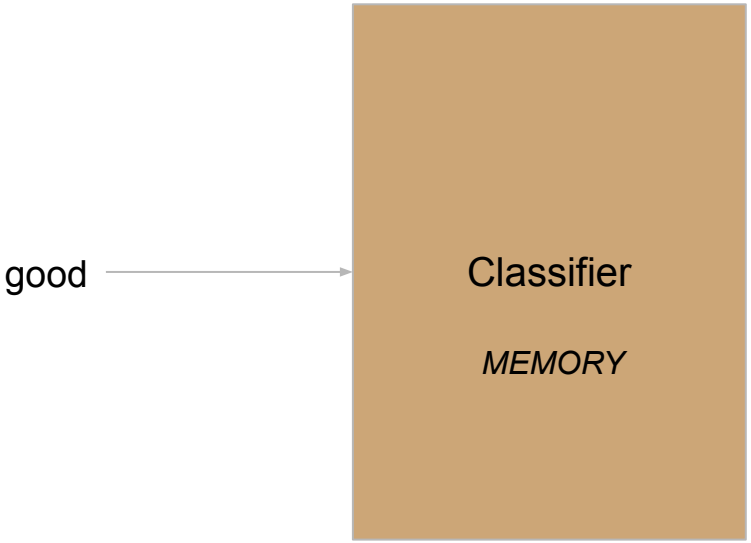

Important context information are saved in memory as we feed one word at a time.

# Approach



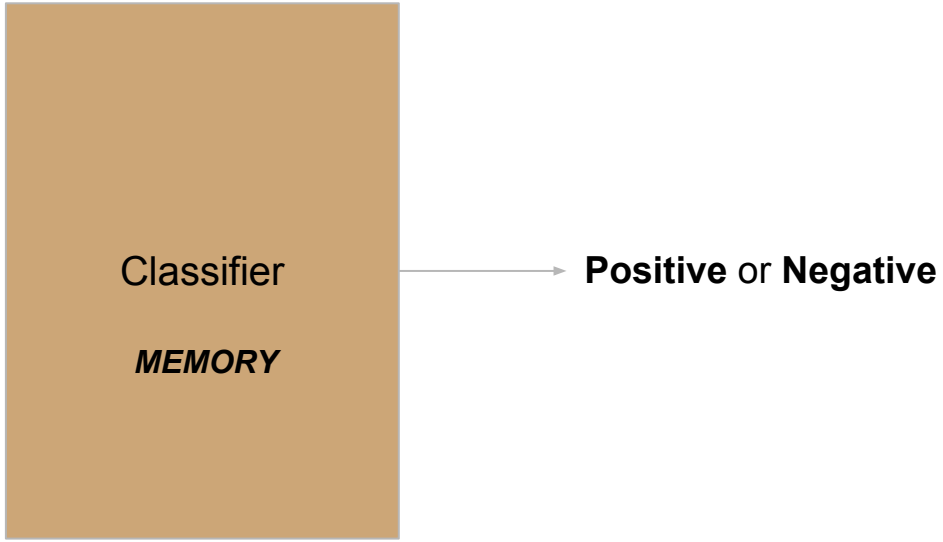

Important context information are saved in memory as we feed one word at a time.

# Approach



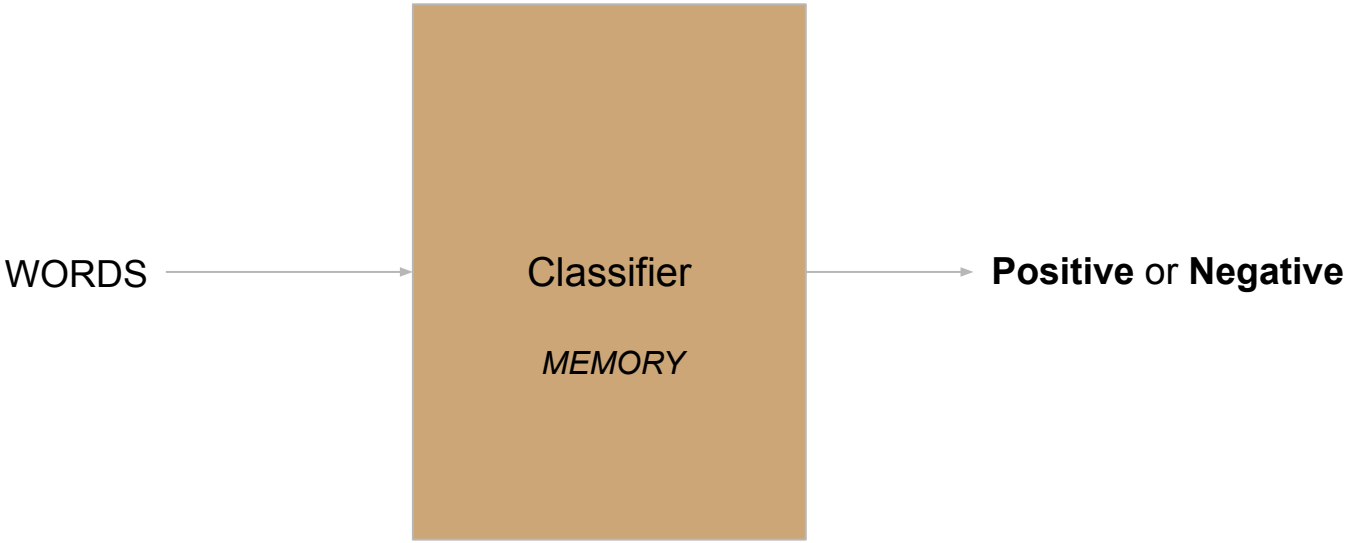

Important context information are saved in memory as we feed one word at a time.

# Approach

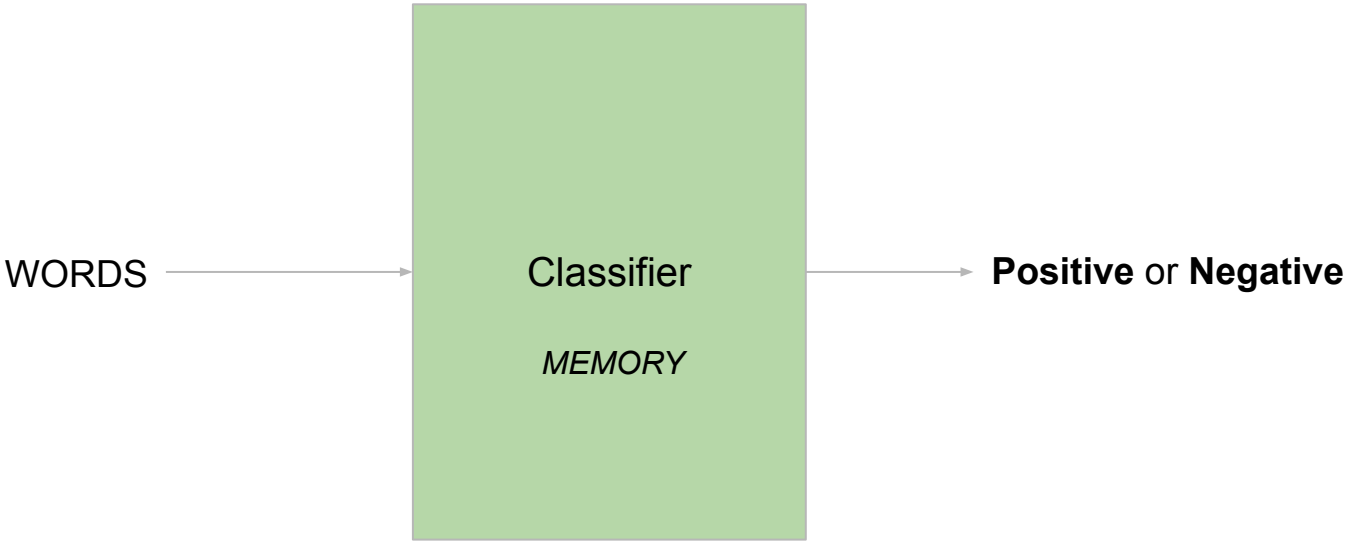



The memory information is used together with input to generate the output.

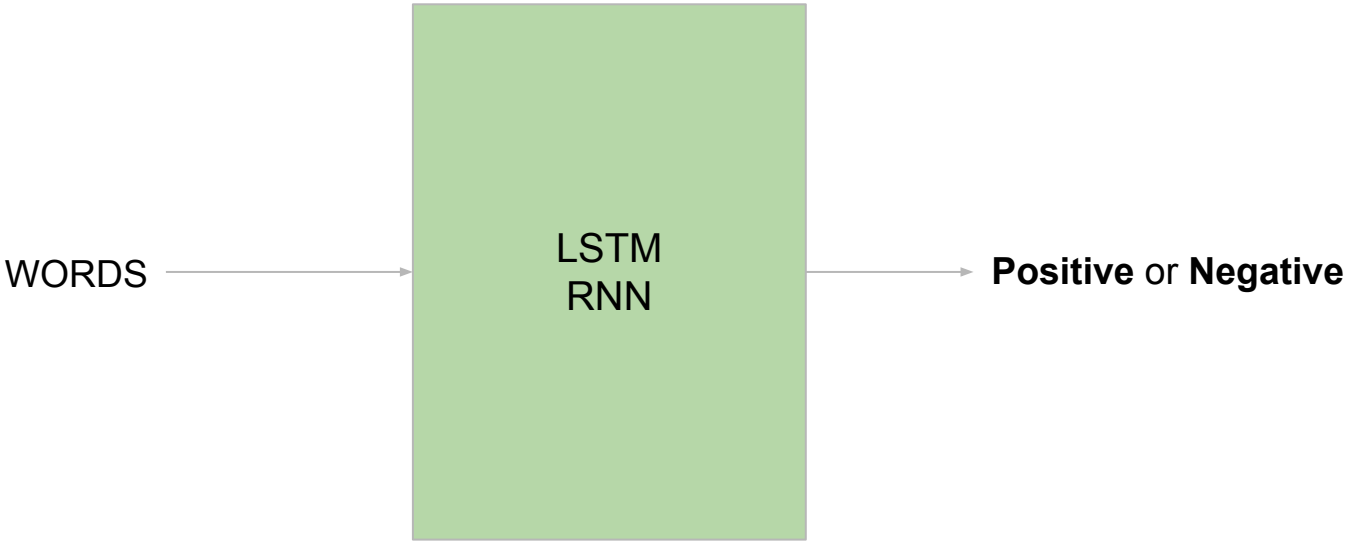
# Approach



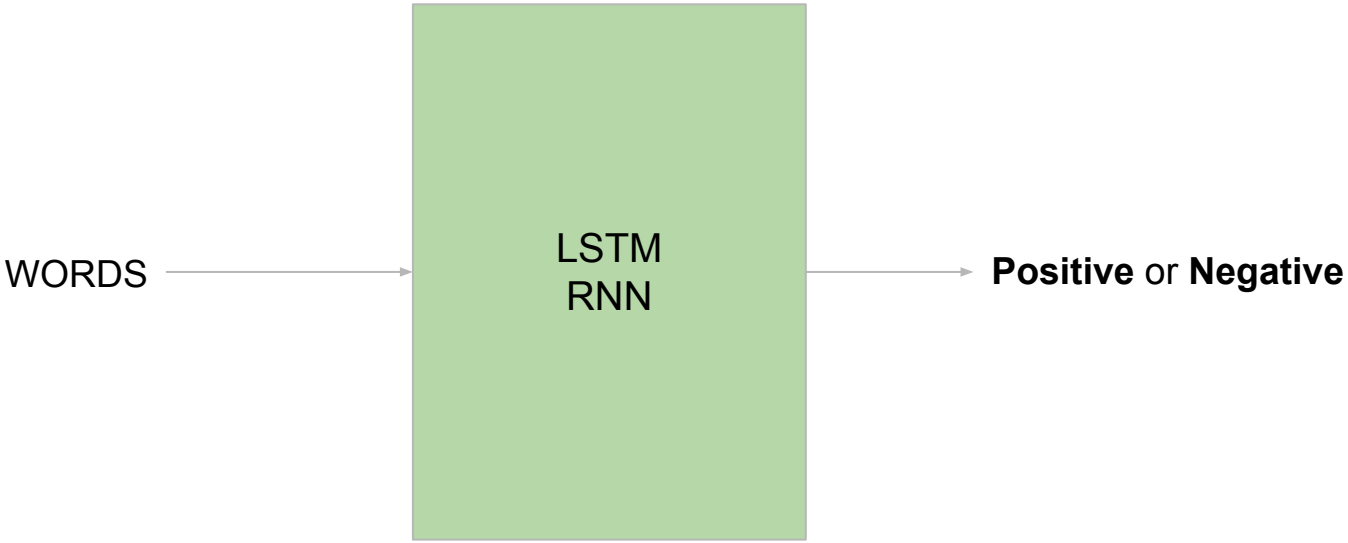
# Approach



# Approach



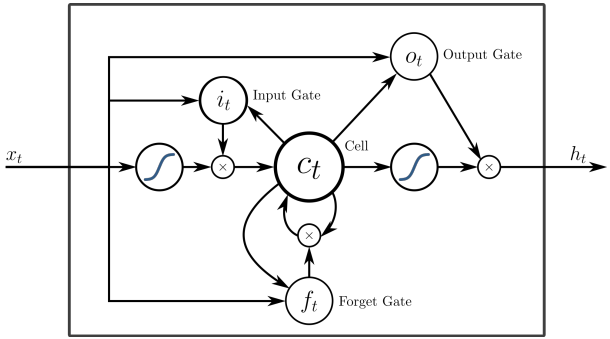
# Approach




*Covered in class but I will go over briefly.*

# LSTM - Review

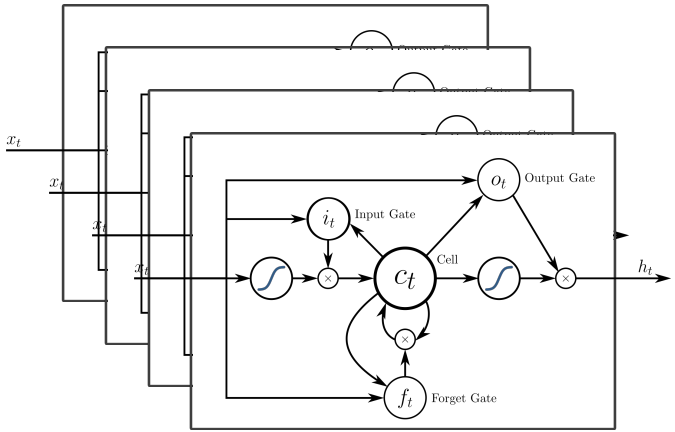
Long Short-Term Memory




An LSTM Cell

*Capable of remembering some context information for some time.*

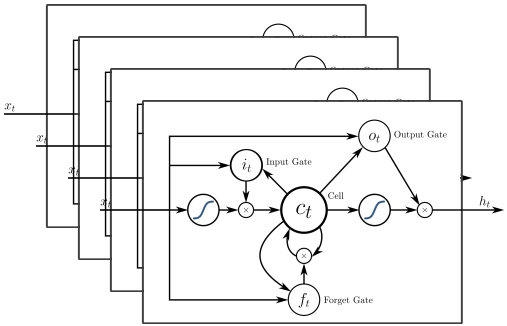
# LSTM - Review

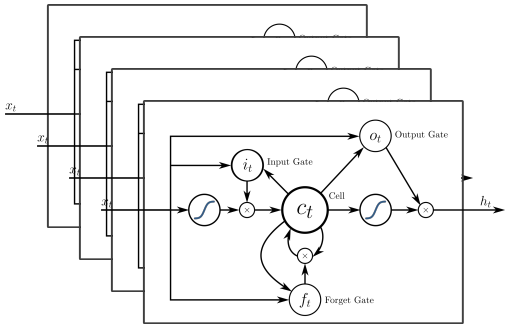


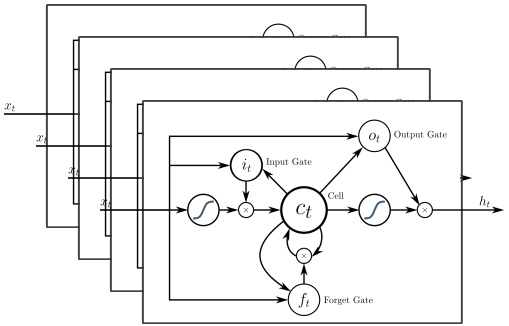

An LSTM Layer

*Capable of remembering multiple context information.*

# LSTM - Review

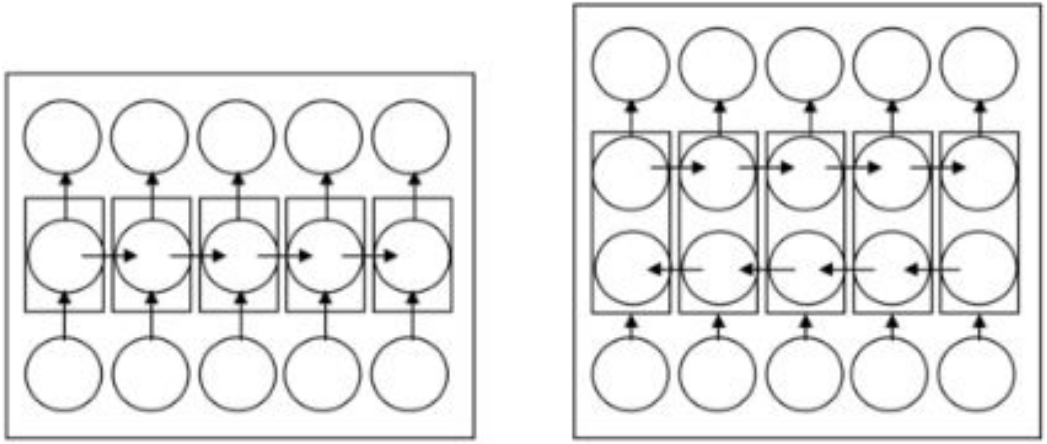



An LSTM Deep Neural Network

# Bidirectional LSTM



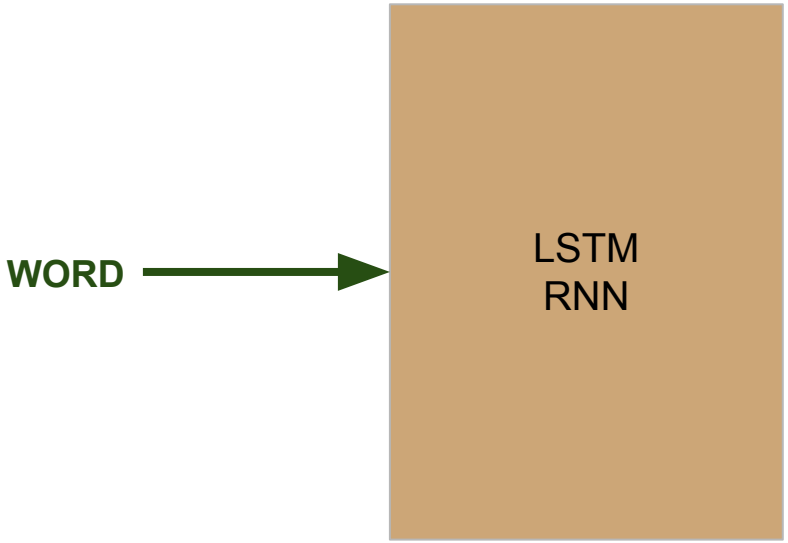
Run input (sequence of words) in two ways:

*Forward direction*

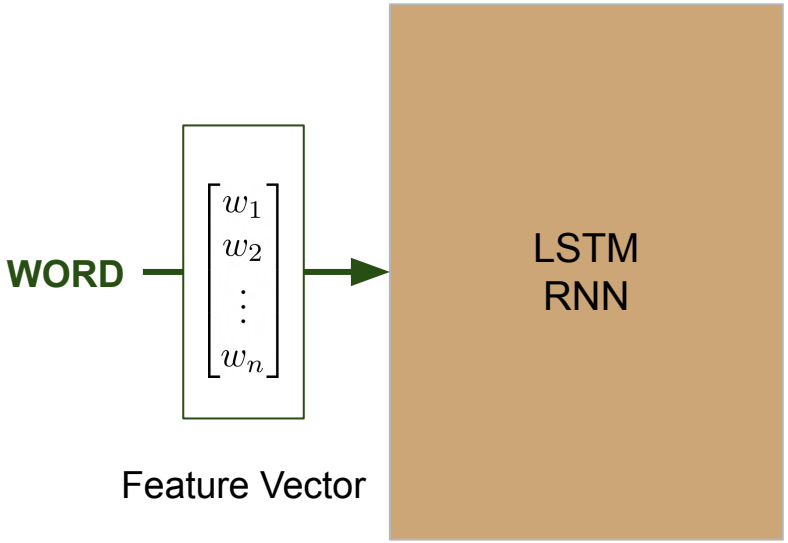
*Backward direction*

Information from both past and future are used for output at any given timestep.

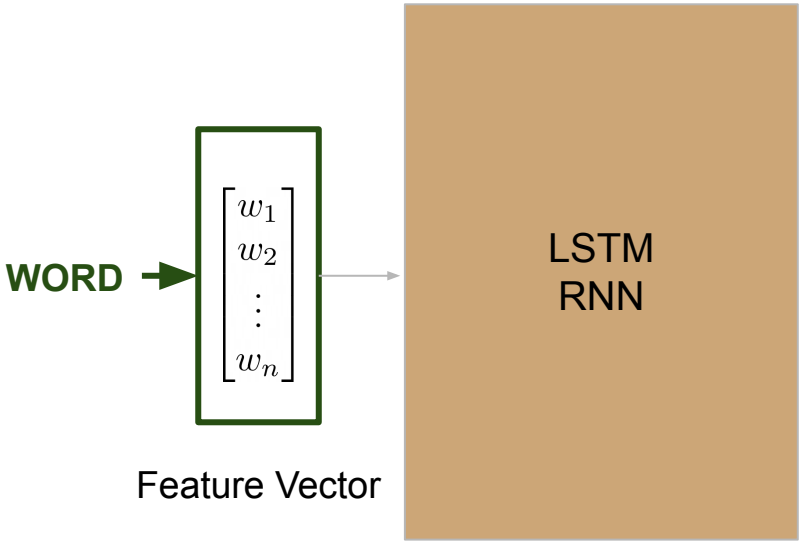
# Word Embedding



# Word Embedding

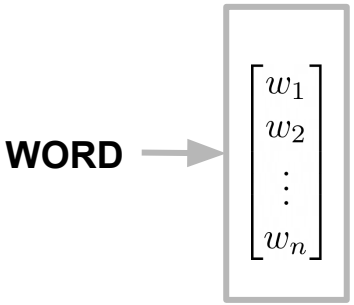


# Word Embedding



# Word Embedding

Word to a Vector



# Word Embedding

Word to a Vector

We want to **capture relationships** between words

# Word Embedding

Word to a Vector

We want to **capture relationships** between words

*One-hot vectors simply is not good enough.*

# Word Embedding

Word to a Vector

We want to **capture relationships** between words

*Better Examples: Word2Vec, GloVe*

# Word Embedding

Word to a Vector

We want to **capture relationships** between words

*Better Examples: Word2Vec, GloVe*

*King is to man what queen is to woman.*

# Word Embedding

Word to a Vector

We want to **capture relationships** between words

*Better Examples: Word2Vec, GloVe*

Vec(King) - Vec(Man) + Vec(Woman) = Vec(Queen)

# GloVe

- Similar to word2vec
  - Trains a large text corpus to find mapping of words to vector
  - But word2vec only considers whether or not two words occur together in the training set
- Also considers multiplicity of co-occurrences
  - How many times a word appear in context of another word in the whole corpus?

# GloVe

$$\text{Similarity between } u_j \text{ and } v_i \propto p_{ij}$$

Conditional probability of word *i* appearing together with word *j*.

$$e^{<u_j, v_i>} = \alpha p_{ij}$$

$$e^{<u_j, v_i>} = \alpha \frac{x_{ij}}{x_i}$$

Number of times word *i* and *j* appear together.

# GloVe

$$\text{Similarity between } u_j \text{ and } v_i \propto p_{ij}$$

Conditional probability of word *i* appearing together with word *j*.

$$e^{<u_j, v_i>} = \alpha p_{ij}$$

$$e^{<u_j, v_i>} = \alpha \frac{x_{ij}}{x_i}$$

Number of times word *i* and *j* appear together.

$$< u_j, v_i > - log\alpha = logx_{ij} - logx_i$$

# GloVe

$$< u_j, v_i > -log\alpha = logx_{ij} - logx_i$$

$$\text{Minimize} \quad (< u_j, v_i > + bias\ terms - logx_{ij})^2$$

$$\text{Minimize} \quad (< u_j, v_i > + b_i + c_j - logx_{ij})^2$$

*Square Loss Function*

# GloVe

$$\text{Minimize} \quad \sum_{i \in V} \sum_{j \in V} h(x_{ij})(< u_j, v_i > + b_i + c_j - log x_{ij})^2$$

*Square Loss Function*

$h(x_{ij})$ is monotonic with $x_{ij}$ in the range $[0, 1]$

*Weight function:*
*To handle the fact that* ***rare or noise co-occurrences are less important than frequent co-occurrences****.*
*Also to ignore numeric instability when* $\boldsymbol{x_{ij} = 0}$*.*

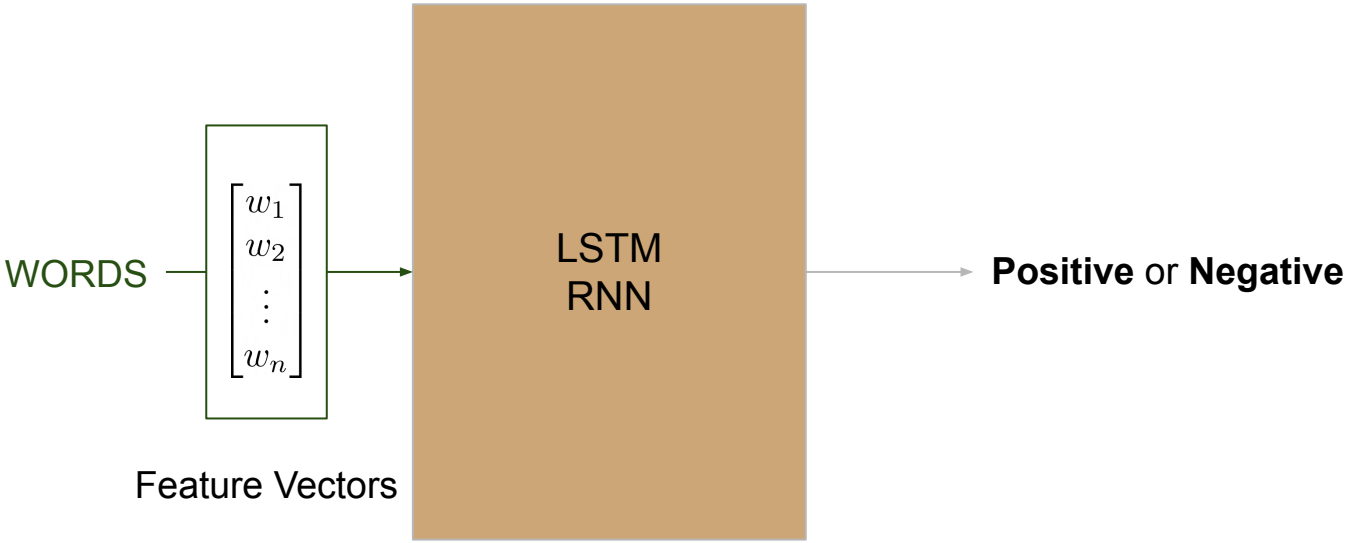
# GloVe

$$\text{Minimize} \quad \sum_{i \in V} \sum_{j \in V} h(x_{ij})(< u_j, v_i > + b_i + c_j - log x_{ij})^2$$

*Square Loss Function*

*My slides oversimplified the maths to give basic idea.*
*See the paper:* https://nlp.stanford.edu/pubs/glove.pdf *to get the full idea.*

# Architecture



# Architecture

# Architecture


**GloVe** Word Vectors

Our training set is too small, so we take pre-trained word vectors from Stanford's 6B dataset.
Vector size = 100

# Architecture


**Bidirectional LSTM**

| | |
|---|---|
| Number of Layers | = 2 |
| Number of Hidden States | = 100 |
| Input Vector Size | = 100 |

# Architecture


**Linear Network**
Input = (Hidden States of initial and final timesteps of the Bidirectional LSTM Layers)
Input Size = 4 * 100
Output Size = 2 (One for positive, another for negative)

# Architecture


**Output**
Two neurons: (Negative) and (Positive)
Whichever is greater.
*Softmax function if we want probability*

**Loss Function**
Cross Entropy based on softmax output

# Training


**Loss Function**
Cross Entropy based on softmax output

**Learning Rate**
0.01

**Number of Epochs**
10

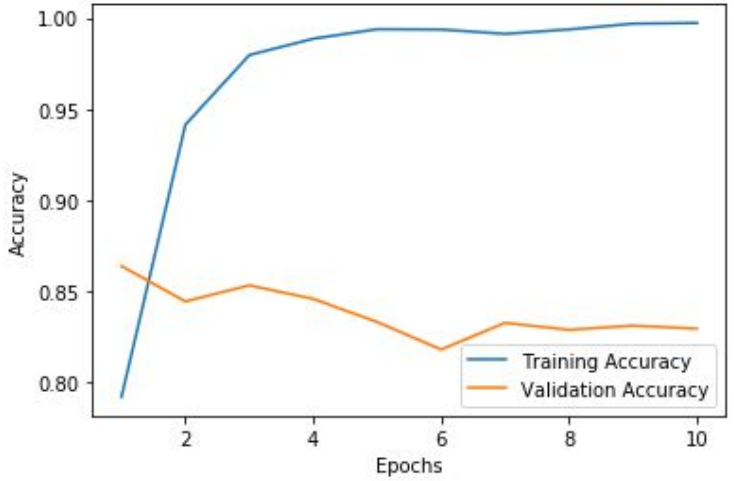
**Batch Size**
64

# Training Results

**Loss Function**
Cross Entropy based on softmax output

**Learning Rate**
0.01

**Number of Epochs**
10




*Accuracy on the Test Set:* ***82.42%***

Let’s look at the code …

THANK YOU.