SENTIMENT ANALYSIS

Hirish Chandrasekaran, Kevin Zhang, Katie Huynh,
Isha Gokhale, Priyasha Agarwal

SENTIMENT ANALYSIS

- **Sentiment analysis**
    - Uses natural language processing and text analysis in order to quantify and learn more about subjective information
    - Requires taking extent corpi of data and classifying parts of it into a pre-defined sentiment
    - Needs training of a model in order to classify the data into categories of real/fake
- **Machine Learning**
    - the use of statistics to find patterns in data
    - an approach to data analysis that involves the building and adapting of models
- **Natural Language Processing (NLP)**
    - an application of machine learning involving the interaction between computers and human language
        - speech recognition, natural language understanding, and natural language generation
    - Refers to analyzing the human language and making sense out of it
    - Focuses on developing models and applications in order to do so
- **Why?**
    - Often times, there are cluttered sources of data online and it is hard to determine what news is real or fake
    - Companies often need to analyze language and its sentiments as well in order to gather data to understand their environment and consumers

LABELS & FEATURES & DATASET

- **Label**
  - Output received from the model after it is trained
  - Whether specific data is real or fake
- **Feature**
  - Property of your training data
  - Each word has a different weight which gets passed through filters in the neural networks
- **Descriptive Statistics of the Data Set**
  - Greatest feature: “said”
  - n-gram approach
    - Best cross-validation score: 94%

APPROACHES TO PREPROCESSING

- **Preprocessing**
  - ***Tokenization***
    - Process of converting text into tokens and then transforming them into vectors using Torchtext
    - Allows us to find which words are not necessary or are "stop" words (contain no sentiment)
  - ***CountVectorizer/TF-IDF*** (term frequency- inverse document frequency)
    - Provides a way to tokenize text and build a vocabulary of words
  - ***Normalization***
    - Makes sure that words with the same meaning are treated equally such as "100" and "one hundred" or "APPLE" and "apple"
    - Main categories: case of the letters, negation (contractions), removing (punctuations, special characters, etc.), lemmatization (finding the base of the word)
  - ***Substitution***
    - Removing HTML markup from the words
- **Train-Data**
  - an array of dictionaries with keys
  - Train function
    - iterates over all examples, one batch at a time
- **Building a vocabulary**
  - every unique word in the data set has a corresponding index

APPROACHES TO PRODUCING AN ACCURACY SCORE

- Finding a way to train and separate/classify data
  - Naive Bayes
  - Support Vector Classifier
- Deep Learning Introduction
  - Torchtext and Pytorch
- Building the machine learning model
  - RNN
    - Simple Model and Upgraded using bidirectionality, lstm, and multilayer
  - CNN
    - Convolutional Neural Network

STAGE I:

scikit-learn, Naive Bayesian approach, and Support Vector Classifier

SCIKIT-LEARN

- Started off the project reading Introduction to Machine Learning with Python
- Software machine learning library
- Uses the NumPy (contains objects that are arrays) and SciPy (contains objects that are matrices) packages
- Worked with pandas (each column can be different types and it allows easier access to CSV files)

NAIVE BAYES

$$\mathbb{P}(y_i|x_1, x_2, \ldots, x_n) = \mathbb{P}(x_1|y_i)\mathbb{P}(x_2|y_i) \times \ldots \times \mathbb{P}(x_n|y_i)\mathbb{P}(y_i)$$

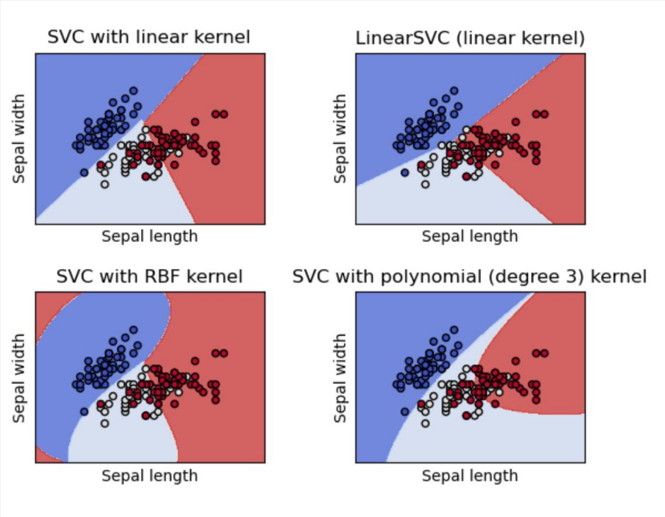
- Independent Conditional Probability Model
- Many Naive Bayes Classifiers (Gaussian, Bernoulli, Multinomial):
  - Bernoulli was for Binary data (counts every time a feature of a class is not zero)
  - Multinomial was for Categorical data (takes into account the average value of each feature)
  - We ended up using the Multinomial Naive Bayes due to the categories of real and fake
- Required a certain "count" of data
- MultinomialNB classifier
  - Uses alpha to control model complexity
  - Larger alpha means smoother model
- Pros: Faster in training
- Cons: Slightly worse generalization performance

Calculations use the Bayes theorem (with the assumptions that the random variables are independent)

Largest probability is chosen as the classification (Maximum a Posteriori - Decision Rule)

SUPPORT VECTOR CLASSIFIER

- an algorithm that takes input data and outputs a line that separates those classes (if possible)
  - **Support vectors** - small subset of training samples that determines the decision function
  - select a hyperplane with max possible margin between support vectors in the given dataset
- **SVM Kernels**
  - Linear
  - Polynomial
  - Radial Basis Function
- **Tuning Parameters**
  - C
    - controls between smooth decision boundary and classifying training points
    - larger value → more intricate decision curves
  - Gamma
    - determines the influence of a training example
    - larger value → every point has close reach



STAGE 2:

PyTorch and Torch Text

PYTORCH

- A deep learning library that allows you to implement neural networks
- Allows you to pick and choose whatever layers you want and their dimension
- Basically the alternative to TensorFlow

TORCH TEXT

- Package with data processing facilities
- Main concept is a Field that defines how data is processed
  - TEXT field determines how data is processed
    - "tokensize = 'spacy'" means that the data will be split based on the spacy model that needs to be downloaded in
  - LABEL field processes the sentiment
- Supports common datasets used in Natural Language Processing

STAGE 3:

CNN and RNN

WHAT IS A NEURAL NET AND TYPES OF NEURAL NETS

Deep Feedforward Neural Network:

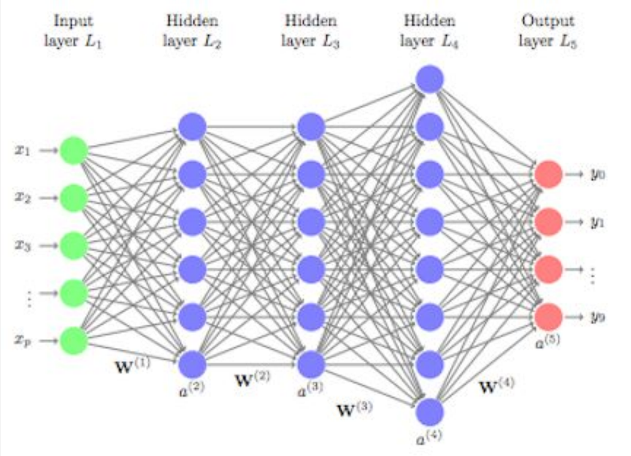
- Also called MLP network, most basic neural network out there
- Essentially inputs go in "one-direction"
- The inputs pass through a series of linear weights and biases and pass through an activation function that tightens the output at each node

Convolutional Neural Network

- We also have Cnn's these are of main research interest in the 21st century
- Not much different from a DFNN except for the fact that the inputs pass through a convolutional layer first

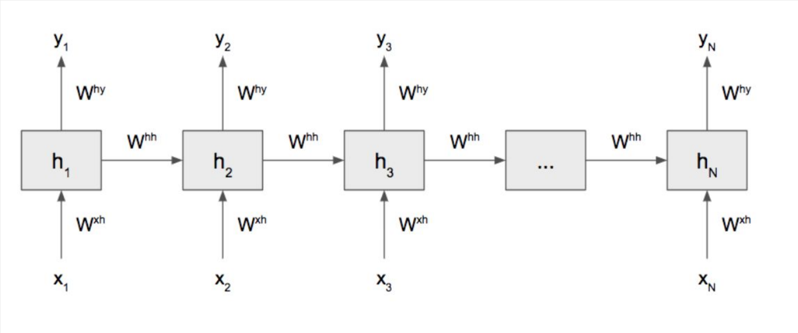
Recurrent Neural Network

- Connections between nodes form a sort of temporal structure, useful for sequential data
- Allow a previous output to be used as an input



RNN

- **In general:**
  - used for sequence data, such as text & speech
  - Feed in an input x_t at some time t to get a hidden state h_t
    - produces a y_t
- **Recurrence** - inputs later in the sequence depend on inputs that were produced earlier in the sequence
  - this is due to the hidden state
  - brings in the idea of short term memory
- **Our project:**
  - uses RNN to recurrently feed in current words x_t and the hidden state from the previous word, to calculate the next hidden state for the text of each news article
    - Long Short-Term Memory (LSTM)



A General RNN

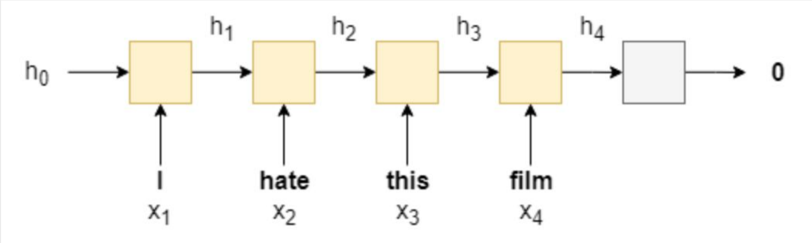
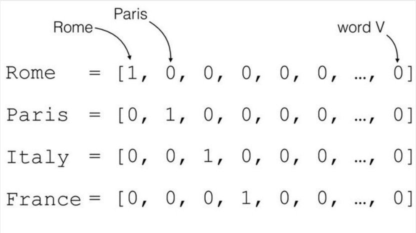


image from pythonmachinelearning.pro

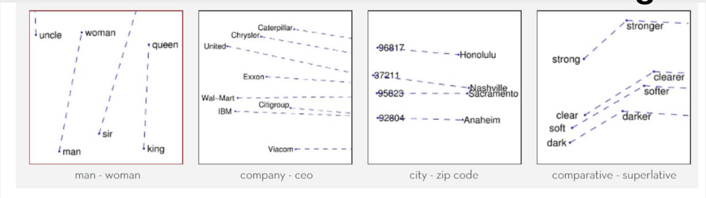
IMPLEMENTATION OF RNN - SIMPLE MODEL

- Layers of the module
  - Embedding layers
    - transform a sparse one-hot vector to a dense embedding vector of sentences
      - dense embedding vector positioned so it is close to other words with similar meaning
  - RNN Layer
    - calculates the hidden states and uses them to calculate a final hidden state that is dependent on previous words
  - Linear layer
    - hidden state goes through the linear layer to produce a prediction (0 for fake, 1 for real)

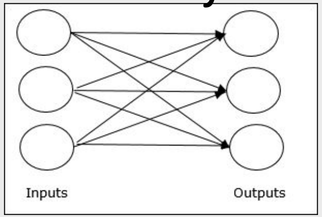
One Hot Vector



Word Embedding



Linear Layer



Images from tutorialspoint.com, towardsdatascience.com

RESULTS OF SIMPLE RNN

Accuracy Score:

- we got an accuracy score of *52.15%*
- this is NOT that good
  - but it is *expected* because a simple RNN suffers from the *vanishing gradient problem*
    - basically, words from the beginning of the sentence have less weight on the final hidden state

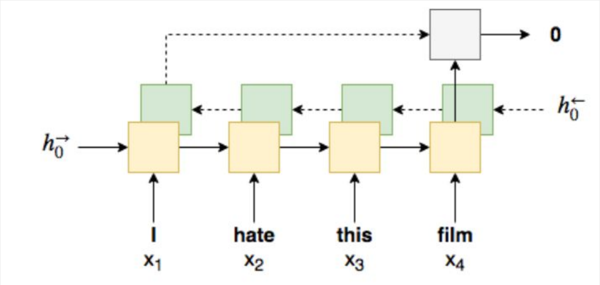
Solution:

- to fix this, we implemented a *Long Short Term Memory (LSTM)* RNN model
  - this model uses an additional recurrent state called cell
    - which uses multiple gates to control the flow of information, into and out of this cell
      - this gives it the name Long Short Term Memory
- We end up with an accuracy of 96.79%

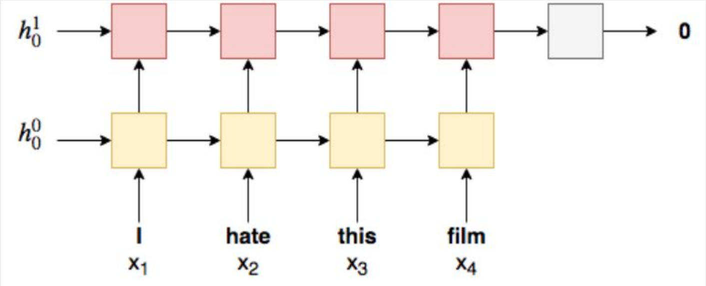
IMPLEMENTATION OF RNN - UPGRADED MODEL

- Long Short-Term Memory (LSTM)
  - returns the output and a tuple of the final `hidden` state and the final `cell` state
  - has an extra recurrent state called cell for long term memory
- Also added Bidirectionality
  - has one going in a normal direction and one in reverse direction
    - so first time stamp is first word and last word
  - make our prediction by concatenating last hidden state of forward and backward RNN
- Also implemented Multi-layer RNN
  - add additional RNNs on top
  - hidden state from lower layer is input to the RNN layer above it

Bidirectional RNN

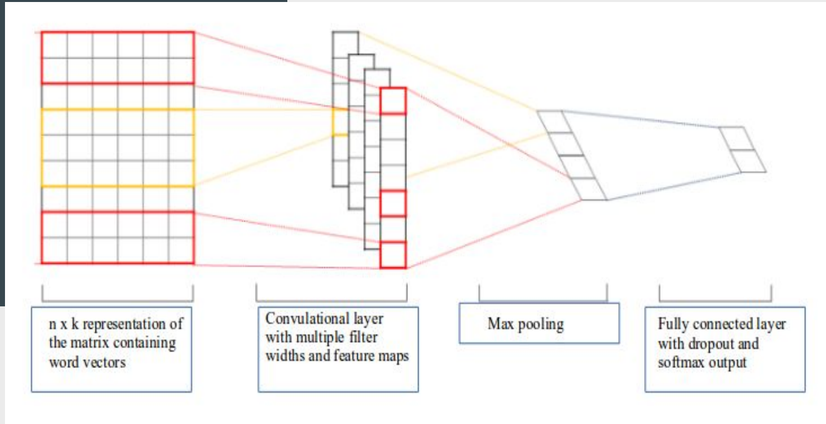


Multi-layer RNN



Images from
https://github.com/bentrevett/pytorch-sentiment-analysis

LAYERS OF THE CNN MODEL



Embedding layer:

- Exactly the same as the embedding layer used in the RNN, we are simply taking the one-hot vector and translating it into a smaller dimensional input

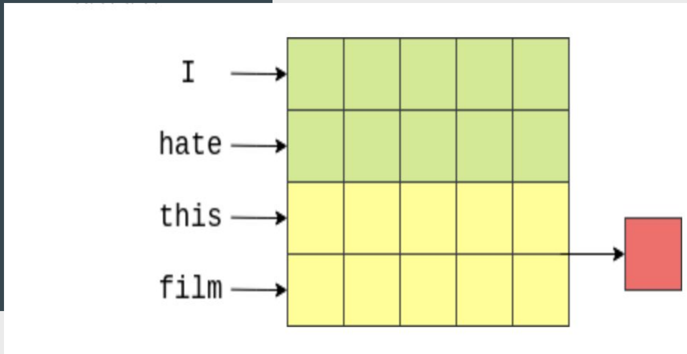
Convolutional Layer:

- Passes multiple filters (with unique weights) over the text that produce a number
- The filters output the maximum number from each filter dimension which is then passed into the linear layer

Linear Layer:

- Evaluates the weight of all the values produces from the filters and gives a final decision

CNN: CONVOLUTIONAL LAYER



- Convolutional Layer
  - We first transform our text data into a sort of image representation
  - We have a 2D image with one channel, the y-axis corresponds to each word, the x-axis corresponds to the dense embedding vector
  - We then apply our convolutional layer shown in yellow to feed into a max pooling layer (simply takes the maximum of a set number of filters)
  - We have different filters that we use for this purpose then pool them together

Image from https://github.com/bentrevett/pytorch-sentiment-analysis

CNN: LINEAR LAYER AND RESULT

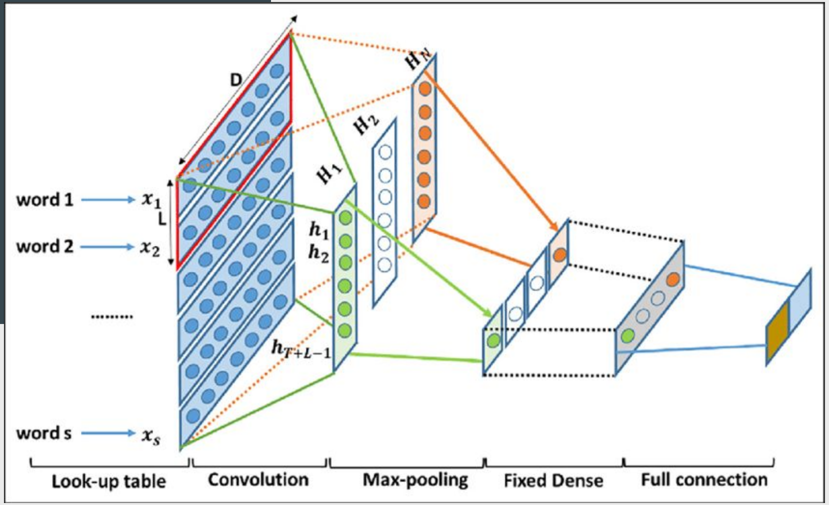


Image from https://www.researchgate.net/figure/Standard-CNN-on-text-classification_fig1_333752473

- Used max pooling to find the maximum value over a dimension
- Our model has 100 filters of 3 different sizes = 300 n grams which are concatenated into a single vector and sent into a linear layer in order to predict the sentiment.
    - The linear layer considers the weight of the 300 n grams and produces a final decision
- Our model produced an accuracy score of 99.83 %
- The particularly low loss number and high accuracy score indicates that our model is reliable.

THANKS

Does anyone have any questions?