PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
POLYTECHNIC SCHOOL
COMPUTER SCIENCE UNDERGRADUATE PROGRAM

AUTOMATING NEWS SUMMARIZATION WITH DEEP LEARNING

MAURICIO STEINERT

Final Undergraduate Work II submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fullfillment of the requirements for the degree of Bachelor in Computer Science.

Advisor: Prof. Felipe Rech Meneguzzi

Porto Alegre
2018

I dedicate this work to everyone who dreamed, walked and ran along with me so it could become reality.

“There is a difference between wanting and doing.”
(Flávio Roberto Gonçalves, Instituto de Artes da UFRGS)

ACKNOWLEDGMENTS

First I would like to thank my family for all support, specially to my wife and son for their patience during all this year. It was not easy, I know.

To my advisor for all support, opportunities, patience and for believing in my work.

To my colleagues at university that share the same drama, for their words of encouragement and support, and laughs when we even know if it is still possible to laugh.

To my colleagues at PROCEMPA, for their understanding, respect and support during this time.

AUTOMATING NEWS SUMMARIZATION WITH DEEP LEARNING

ABSTRACT

A text summary consist of a short version of a text that retain its key aspects. Automating test summarization can help users to navigate through large volumes of information without wasting too much time. In this work, we develop methods to automate text summarization in the news domain, specifically of news available on the Internet, using techniques in the field of Artificial Intelligence and Natural Language Processing. We developed three methods, the first based on Feed-forward Neural Networks, the second using Recurrent Neural Networks, and a third using word vector arithmetic.

Keywords: automatic text summarization, artificial intelligence, neural networks, natural language processing, information retrieval.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN. – Convolutional Neural Network

MLE. – Maximum Likelihood Estimation

MSE. – Mean Squared Error

NLP. – Natural Language Processing

ReLU. – Rectified Linear Unit

RNN. – Recurrent Neural Network

CONTENTS

1. INTRODUCTION

Automatic text summarization produces a summary, i.e., a short length version of the source text that contains their most relevant content. As stated by Gambhir [3], text summarization is a challenging task that must deal with issues like redundancy, temporal dimension and other aspects related to text writing.

Nowadays we are overwhelmed with large volumes of information. The Internet give us access to many data of the most distinct types like texts, images and videos in an ever growing trend. The advent of Internet of Things put every type of device to generate even more data, from telemetry sensors that monitor whole cities to health care gear that watch over a subject. Personal devices like smartphones also generate a lot of data. Having access to all this data is useless if its not possible to quickly evaluate its content and extract relevant information from them. Summaries allow users to navigate through a large volume of data quickly and search for specific information.

Manually generating summaries, however, is also a time-consuming task that can be influenced by writer's background on the matter due to his personal opinion reflected into the summary [10]. Automating text summarization is time saving for writers and yield more objective results without taking into account subjectivity inherent to human beings.

The first automatic text summarization attempt dates from 1958 [10] motivated by the growing number of academic articles available at that time. Recent examples are news websites, mailboxes full of messages and results from search engines. Over the time different methods and approaches have been developed in this field of research, from probabilistic models [10] to complex neural networks [13] based solutions.

Our overall goal is to develop a learning-based method for text summarization for the online news domain. Specifically, we aim to summarize texts with an average of 800 words for a full news article, where the corpus we choose is a set of news extracted from CNN/Dailymail detailed in Section 6.1.

During first semester, we evaluated a set of academic researches, where two of them give us an overview on the matter: Gambhir [3] and Kahn [8]. These articles helped us to identify key concepts and served as starting point to build a solid background about our problem.

We also investigated related work, where we identified that different methods have been used to solve this problem, from statistical models that use word frequency to identify most significative words or sentences to complex neural networks trained over large volume of texts. Neural networks are reported to achieve better results when compared to other techniques.

During this period, we practiced with word embeddings to represent words and sentences. The result of these experiments is the Vector Offset Summarization method, detailed in Chapter 9.

After research stage, we start implementing our first model based on a Feed-Forward Neural Network for extractive summarization and later, we implemented a Recurrent Neural Network solution that is specialized in sequence data processing.

This work is structured as follows:

- Chapter 2 introduces key concepts about Natural Language Processing and language models, text summarization task and automatic evaluation metrics.
- Chapter 3 provides an overview about Machine Learning concepts and components.
- Chapter 4 introduces Neural Networks concepts, network architectures and applications.
- Chapter 5 details neural networks applied to language processing tasks.
- Chapter 6 details our data set structure and pre-processing task, word vectors generation and development tools.
- Chapter 7 details using Feed-Forward Neural Networks in text summarization task, where we trained and explored a set of design choices and evaluated the corresponding results.
- Chapter 8 details using Recurrent Neural Networks in text summarization task, where we trained a neural network to predict scores for each sentence in texts.
- Chapter 9 details the Vector Offset Summarization method, how it is implemented, its features and results.
- Chapter 10 evaluates results by manually examining examples extracted from data set and comparing attained results with ground-truth values. We also interpret results from an human point of view.
- Chapter 11 discuss related work, examine some other researches and compare its results with ours.
- Chapter 12 discuss our conclusions after evaluating our developed methods and forecast further research possibilities.

2. NATURAL LANGUAGE PROCESSING AND TEXT SUMMARIZATION

Natural Language Processing is a class of algorithms that allow computers to interpret natural languages used in human communications. The main reasons we want computers to understand natural languages is to communicate with humans and to acquire information from written language [16].

Computers work based on formal languages with a well-defined structure: a set of valid symbols known as vocabulary, a grammar that defines the rules for communication, and semantic that defines the meaning associated to sentences. These features allow the generation of an infinite number of sentences and make formal languages free of ambiguity.

Natural languages, on the other hand, don't have a well-defined structure, allowing two sentences with different structures to have the same semantic meaning. Another difficulty is that natural languages are very large and keep changing. With these difficulties in mind, we must face our models as an approximation of the language that cannot achieve perfect results at every situation.

2.1 N-gram Language Model

Sentences in natural languages contain words and other set of symbols such as punctuation and special characters: words are composed of characters, sentences are composed of sentences, and so on. *N*-gram is a language model that deals with the probability distribution over sequences of *n* characters, syllables, words, sentences and other units, where the length of *n* symbols is called a *n*-gram. Common configurations of *n*-gram models include "unigrams" when $n = 1$, "bigrams" when $n = 2$ and "trigrams" when $n = 3$.

The probability distribution P for a sequence of N symbols or words c_i can be evaluated by taking into account only the preceding symbols or word probabilities, as shown in Equation 2.1:

$$P(c_1 : N) = \prod_{i=1}^{N} P(c_i | c_1 : c_{i-1}) \tag{2.1}$$

N-gram model is applied to problems like text classification. In this case, texts are stored into a structure feature/value, where feature are words in text and value is its frequency in text. This feature/value structure is called Bag of Words [5].

2.2 Neural Network Language Model

Neural Network Language Models represent words as a high dimensional vector composed of real values that is capable of capturing meaningful syntactic and semantic regularities of words [11]. This word vector is also called word embedding.

This model generalizes data better than *n*-gram models, because while *n*-grams models work with discrete unrelated units, word vectors that represent similar words are likely to have similar vectors. This characteristic allow us to evaluate similarity between words through arithmetic operations. A common example is the analogy shown in Equation 2.2:

$$x_{king} - x_{man} + x_{woman} \approx x_{queen} \tag{2.2}$$

As shown in Figure 2.1, the vector between x_{man} and x_{woman} have approximately the same length as x_{king} and x_{queen}. Intuitively, when we extract x_{man} from x_{king} we keep a feature that we can call informally *royalty*, without an associated *gender* feature. When we sum x_{woman} to this feature, adding the *gender* feature, the resulting word vector is close to x_{queen}.

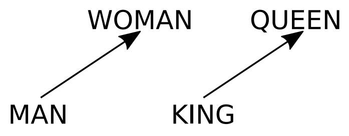


Figure 2.1 – Words relationship represented as vectors.

Word vectors are derived from machine learning processes, which requires substantial training data to generate a meaningful representation of words and their relationships. This process is detailed in Section 5.1.

2.3 Text summarization

Text summarization consist of generating a shorter version of a document while retaining its key concepts [3]. Automatic text summarization dates from the 1950s [10] and along the years many methods have been developed to solve this problem that lies in the field of Natural Language Processing and Artificial Intelligence. A good summary must offer coverage of information, information significance, avoid redundancy in information and present cohesion in text [3].

Some issues that need to be addressed include redundancy (avoiding repeating the same idea), temporal dimension (new information that overrides earlier information, a common situation in multi-document summarization that present the same event at distinct times), co-reference (relate distinct sentences to the same information) [4], which makes this a complex task to accomplish.

Generating a summary involve selecting the most representative features in a text. The following text features can be used to select key sentences [8]:

- Term Frequency, a statistic metric that provide salient terms based on term frequency.
- Location of words, relying on the intuition that important sentences are located at certain positions in text, such as beginning or end of a paragraph.
- Cue Method, where words have positive or negative effect in a sentence weight indicate significance or key idea (for example, expressions like "in summary", "in conclusion", the paper describes", "significantly").
- Title/Headline words, assuming that words that appear in these sections of a document and occurs in sentences are positively relevant to summarization.
- Sentence length, where short sentences express less information, and very long sentences are also not appropriate for summary.
- Similarity between a sentence and the rest of the document.
- Proper nouns, where sentences that contain proper nouns are considered important to summary.
- Proximity, where the distance between text units determine a relationship between them.

Automatic text summarization resembles question-answering algorithms because both return a sentence, phrases or simple words as output. Instead, automatic text summarization does not require a query to be provided: the algorithm must evaluate the whole text, identify and return the most relevant features in it.

Automatic text summarization methods can be broadly classified as extractive summarization methods, where the summary is generated by selecting and copying texts' most relevant sentences and outputting it as summary, or abstractive summarization methods, where the algorithm interprets the text and write a summary from scratch, avoiding reusing words of original text.

Extractive summarization methods can be classified as follows [3]:

- Statistical based approaches use statistical information like frequency, position and length of sentences to measure its importance. Each sentence receives a score that is used to evaluate its relevance. This approach is language independent and do not require any additional linguistic knowledge.

Statistical approaches usually use Bayes' rule (Equation 2.3) to evaluate the probability of a text unit (for example a sentence s) belong to a summary S, given F_k features.

$$P(s \in S|F1, F2, ..., F_k) = \frac{P(F1, F2, ..., F_k|s \in S)P(s \in S)}{P(F1, F2, ..., F_k)} \tag{2.3}$$

- Topic based approaches generates a summary of main topics, where a topic structure is defined by topic themes that occurs frequently in a collection of documents. This approach is usually applied to multi-document summarization.
- Graph based approaches structure words and sentences as nodes with edges establishing relationship between them.
- Discourse based approaches represent connection between sentences and parts in a text, assuming that coherent texts possess some kind of relation between its various parts.
- Machine learning based approaches learn from data. Given a large corpus of text as input with their respective summaries, the machine can learn to generate summaries based on them. If the input data doesn't contain the summaries, the machine learning algorithm try to discover hidden structures and relations by itself without previous training.

Abstractive summarization identifies the main concepts and ideas and re-interprets them, without copying sentences from the source text. This method demands extensive natural language processing and is more complex than extractive methods but produces highly coherent, cohesive, information rich and less redundant summary.

Abstractive summarization methods are broadly classified into two categories: Structured based approach and Semantic based approach.

Structured based approaches encode the most important information from the document through cognitive schemas such as templates, extraction rules and other structures, for example [8]:

- Tree based methods use a dependency tree to represent the text of a document and different algorithms are applied for content selection for summary.
- Template based methods use a template to represent the whole document, and linguistic patterns or extraction rules are applied to identify text snippets that will be mapped into template.

- Ontology based methods use a knowledge base to improve the summarization process, assuming the intuition that a set of documents is domain related because deal with the same topic or event, where each domain have his own knowledge structure that can be better represented by ontology. This method requires the intervention of a domain expert to define the knowledge base, which is a time-consuming task.
- Lead and body phrase methods aim to rewrite a lead sentence by using operations of insertion and substitution of phrases that have the same head chunk in the lead and body sentences.
- Rule based methods represent documents as a list of categories and a list of aspects, where content selection is based on answers to one or more aspects of a category, followed by a generation pattern that produces the summary.

Semantic based approaches generate a semantic representation of documents to feed into natural language generation. These methods look for verbs phrases and nouns phrases by processing linguistic data. Some approaches and methods are:

- Multimodal semantic model generates a semantic model that represent concepts relationship taking into account contents like text and images of multimodal documents, where important content is rated and expressed as sentences.
- Information item based method generates summary based on abstract representation of documents, where abstraction representation is an Information Item, which is the smallest element of coherent information in a text.
- Semantic Graph Based Method aims to create a semantic graph for the source document, where each node represent a concept and edges represent relationship between them. The method reduces the graph and then generate the final abstractive summary from the reduced semantic graph.

2.4 Text summarization evaluation metrics

Generating manual summaries is a time-consuming task as well as evaluating summaries quality by hand. Automating summary evaluation is necessary and is still an open problem in Natural Language Processing field. A good summary must comply to a set of qualities that includes coherence, conciseness, grammaticality, readability and content [9].

The most traditional solution available for automatic summary evaluation is the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metrics [9]. ROUGE is used in scientific research and contests such as the ones on Document Understanding Conferences as official evaluation metric for text summarization [3].

ROUGE receives as input the generated summary and reference summaries generated by humans and measures similarity between them. It evaluates the number of overlapping units as *n*-gram, word sequences and word pairs compared to human generated summaries.

There are different metrics evaluated by ROUGE:

- ROUGE-N computes n-gram recall between a candidate summary and a set of reference summaries (Equation 2.4). This equation allows the use of a set of reference summaries instead of a single ground-truth summary.

$$\text{ROUGE-N} = \frac{\sum_{S\in\{ReferenceSummaries\}} \sum_{gram_n \in S} count_{match}(gram_n)}{\sum_{S\in\{ReferenceSummaries\}} \sum_{gram_n \in S} count(gram_n)} \tag{2.4}$$

- ROUGE-L refers to the Longest Common Subsequence between generated and reference summaries. Given sequences $z = [z_1, z_2, ..., z_n]$ and $X = [x_1, x_2, ..., x_m]$, if exists an increasing sequence $[i_1, i_2, ..., i_k]$ of indexes of X such that for all $j = 1, 2, ..., k$ we have $X_{i,j} = Z_j$. This metric are based on the intuition that the longer the common sequence of two summaries is, the more similar they are. The Longest Common Subsequence is computed as shown in Equations 2.5 (recall), 2.6 (precision) and 2.7 (F-score). This metric doesn't require consecutive matches.

$$R_{LCS} = \frac{LCS(X, Y)}{m} \tag{2.5}$$

$$P_{LCS} = \frac{LCS(X, Y)}{n} \tag{2.6}$$

$$F_{LCS} = \frac{(1 - \beta^2) R_{LCS} P_{LCS}}{R_{LCS} + \beta^2 P_{LCS}} \tag{2.7}$$

where m is the total of words of reference summary sentence, n is the total of words of candidate summary sentence, β is defined as $\frac{P_{LCS}}{R_{LCS}}$ when $\frac{\partial F_{LCS}}{\partial R_{LCS}} = \frac{\partial F_{LCS}}{\partial P_{LCS}}$, but usually is set to a big number ($\rightarrow \infty$).

- ROUGE-W (Weighted Longest Common Subsequence) is similar to ROUGE-L but takes into account the length of consecutive matches, ROUGE-S (Skip-BGram Co-occurrence Statistics) that measures the overlap of skip-bigrams between a candidate document and a set of candidate documents, and ROUGE-SU that is an extension of ROUGE-S that takes into account unigram as counting unit.

According to Lin [9], ROUGE-2, ROUGE-L, ROUGE-W and ROUGE-S are better for single summaries. For very short text summaries, ROUGE-1, ROUGE-L, ROUGE-W, ROUGE-SU4 and ROUGE-SU9 are usually applied.

To get a better intuition of how ROUGE scores work, Table 2.1 shows some examples of sentence pairs and ROUGE scores comparing *sentence 1* to *sentence 2*. We verified that this evaluation metric take into account *n*-gram similarities, is case sensitive and gives low score values for sentences with distinct *n*-grams but semantic similarities.

		ROUGE		
Sentence 1	Sentence 2	1	2	L
exact sentences get full scores	exact sentences get full scores	0.99	0.99	0.99
word	word	0.99	0.0	0.99
one simple test	another simple test	0.66	0.49	0.66
sentence one	Sentence one	0.49	0.0	0.49
summarization returns a short version of content	summarization techniques generate a shorter interpretation of text	0.39	0.0	0.39
this is a random sentence	something else	0	0	0

Table 2.1 – Examples of ROUGE scores comparing sentence pairs.

Performance metrics are usually computed in terms of precision and recall [5]. Precision reports the fraction of predictions that are correct, while recall reports the fractions of true predictions that are correct. To summarize our performance predictions, we use F-score, as defined in Equation 2.8, where *p* stands for precision score and *r* for recall score.

$$F = \frac{2pr}{p + r} \tag{2.8}$$

3. MACHINE LEARNING

Machine learning is an area of Artificial Intelligence that studies how to make machines learn from raw data [5]. Alternatively, Mitchell [12] defines learning as the capability to get better at a task based on experience as follows: "a computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*".

Machine learning algorithms are currently applied to solve classification problems (identify to which class an input data belongs), regression problems (predict a numerical value given some input), transcription problems (transform an input unstructured data into a discrete representation), machine translation (convert data from a language to another), anomaly detection (identify unusual data given a set of input data), synthesis and sampling (generate new data that is similar to input data), to name a few.

Machine learning algorithms require a data set, a set of input data that the algorithm uses to learn and compute results, a cost function that evaluates the performance of a model by comparing the calculated output with provided input data, an optimization procedure that defines how the learning process occurs, usually by minimizing the cost function, and a model that defines how an input x is mapped to an output $\hat{y}$ [5].

3.1 Data set

A data set is a collection of data that is used to feed a learning algorithm. Usually the dataset is structured into a design matrix $X^{(m,n)}$, where m represent the number of examples and n the number of features per example. Some data sets possess a Y^{m} vector that represent the ground-truth value or labels for each example, but its presence is not mandatory.

Under some situations, data set can assume different configurations, like in Recurrent Neural Network architectures, where the network expect each example configured as a series of time steps, assuming a configuration $X^{(m,t,f)}$, where m represents the number of examples, t the number of time steps per example, and f the number of features per time step.

3.2 Cost function

A cost function evaluate the performance of the learning process by computing the average error between a $\hat{y}$ value generated by the algorithm and a ground-truth value provided for all entries in data set. This average error is computed by a Mean Squared Error (MSE) function (Equation 3.1).

$$MSE = \frac{1}{m}\sum_{i}^{m}(\hat{y} - y)^2 \tag{3.1}$$

Mean Squared Error is helpful to get intuition about cost function. However, in practice its being replaced by Maximum Likelihood Estimation that requires fewer examples to get a better generalization over data [5].

Applied to a learning process, using Maximum Likelihood Estimation we want to minimize the cross-entropy between the training data P_{data} and the model distribution P_{model} (Equation 3.2), where $\mathbb{E}$ is the expected value for a given probabilistic distribution with respect to a function.

$$J(\Theta) = -\mathbb{E}_{x,y\ \hat{P}_{data}} logP_{model}(y|x) \tag{3.2}$$

3.3 Optimization procedure

The objective of an optimization procedure is to minimize a cost function by improving the parameters of the model so the algorithm can best fit a data set. In the case of linear models, we can use Normal Equation (Equation 3.3) that is a closed form solution that minimizes a linear function accurately.

$$\nabla_w MSE = 0 \Rightarrow w = (X^T X)^{-1} X^T y \tag{3.3}$$

where $X \in \mathbb{R}^{mxn}$ contain m entries with n attributes. Each attribute has a coefficient associated to it, where linear functions have x as coefficient. Usually we add a new column b to represent bias with start value $b = 0$ for each entry. $y \in \mathbb{R}^m$ is a vector with labeled values for each entry [5].

Not all models, however, have a closed form to minimize its parameters as linear functions do. In this case, we use iterative procedures to get an approximation of values that best fit the training data set.

Optimizing Mean Squared Error consist of minimizing the error between y and $\hat{y}$. Optimizing Maximum Likelihood Estimation (MLE), on the other hand, consist of maximizing the similarity between a computed model $P_{model}(x, \Theta)$ and $P_{data}(x)$ by adjusting Θ parameter, as shown in Equation 3.4.

$$\Theta_{MLE} = argmax_{\Theta} \sum_{i=1}^{m} logP_{model}(x^{(i)}; \Theta) \tag{3.4}$$

Gradient descent is an iterative optimization procedure that uses partial derivative functions to decide how a parameter must be changed to best fit a data set by evaluating the slope of the function in the search for a minimal value until a stop condition is reached (total number of iterations, relative error threshold, cost function threshold). Equation 3.5 computes one step of gradient descent over an arbitrary model parameter, where ϵ is the learning rate that defines the step size of an iteration.

$$x' = x - \epsilon \nabla_x f(x) \tag{3.5}$$

Stochastic Gradient Descent is an extension of Gradient Descent that, instead of computing the cost function for each example, computes the cost for a small group of examples called mini-batches. This technique aims to reduce the computational cost of calculating derivative functions and cost for each example, taking into account that a good generalization requires a large volume of examples.

Adam is an adaptive optimization algorithm that adjusts the learning rate during training process [5], leading to fast training loss minimization. Gradient values are computed for all parameters, and internal variables that control the algorithm influence how the parameters are updated. These internal variables are adjusted based on previous gradient computation.

3.4 Model

A model defines how an input value x are mapped to an output value $\hat{y}$. A simple example are linear model, which uses a linear function (Equation 3.6), where $w \in \mathbb{R}^n$ is a set of n weights that define how each feature affects the prediction and b is a bias value (Equation 3.6). In other words, this algorithm computes the probability of y given x, or $p(y|x)$.

$$\hat{y} = w^T x + b \tag{3.6}$$

Linear functions are said to have degree or capacity 1 because we deal with only one coefficient. Choosing a model have a direct impact over training and test performance accuracy because the capacity defines the number of coefficients available for adjustments, what can lead to conditions like underfitting and overfitting.

In a machine learning algorithm, when the data set belongs to the same probability distribution, for example a set of academic articles, we expect the training error to be small and the gap between training and test error to be as small as possible. When we are not able to learn a model with a low error value on the training set, we are in a situation called underfitting. If the algorithm have low training error but the gap between training error and test error is too large, we are under a overfitting situation.

The ability of a machine learning algorithm fit a data set with low error is related to the choice of a model that defines how x is mapped to $\hat{y}$. Informally we can attribute a capacity to a model, where models with low capacity cannot fit the training set, resulting in underfitting. Models with high capacity, however, fit well the training set but fail to generalize to test set, resulting in overfitting [5].

Figure 3.1 illustrate two data sets, which values are plotted as red markers, and respective performance measured by Mean Squared Error. The example on the left do not fit perfectly the data set, but get a good approximation. The example on the right, on the other hand, present a high error value, failing to fit the training set and falling into a condition of underfitting.

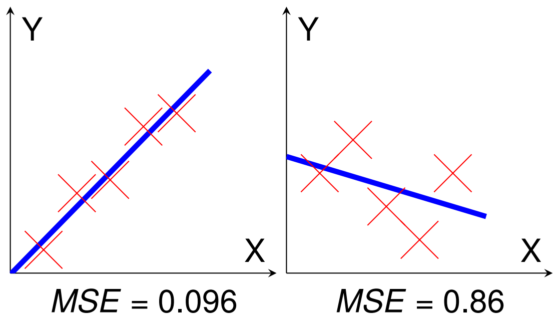


Figure 3.1 – Linear functions fitting two distinct datasets with respective Mean Squared Error.

Underfitting is solved by using a function with higher capability, in other words, by choosing as model a more complex function that can better represent this data set. In practice, this is usually done by using polynomial functions with higher degree.

Figure 3.2 show the same data set presented earlier with different functions minimized to fit it:

- functions of degree 1 and 2 are in underfitting condition.
- function of degree 3 is the one that best match the dataset.
- higher degree functions that fit the dataset extrapolate in-between values are under overfitting condition.

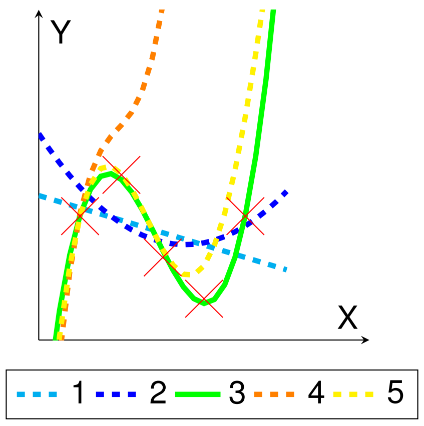


Figure 3.2 – Functions with degree varying in range 1 to 5 optimized to previous data set.

Overfitting condition occurs when the model fit well to training set but is unable to generalize well over validation set and test set. Usually this condition is caused by model capacity higher than necessary to represent the training set.

Regularization methods like weight decay reduces the effect of overfitting condition by computing Mean Squared Error with preference for small weight values, where λ is a hyper parameter (Equation 3.7) that defines how weights are updated. Hyper parameters are parameters that cannot be learned by our learning algorithm during training process but control how it behave.

$$J(w) = MSE_{train} + \lambda w^T w \tag{3.7}$$

3.5 Supervised learning algorithms

Supervised learning is a class of algorithms that receive as input data a pair of values $\{(x^1, y^1), ..., (x^m, y^m)\}$ composed of m entries, where x_i is a vector of features and y is the expected value for that entry, also called ground-truth value or label.

The data set is commonly divided into subsets. A training data set is used during training stage, a validation data set is used for fine-tuning the model, and a test data set to evaluate the generalization capability of models, in other words, the accuracy of the model over unseen data.

Common supervised learning algorithms include linear regression, an algorithm that predicts an output $\hat{y}$ given an input x using a linear function as model. Another algorithm is logistic regression that computes the probability of a given input example x belong to a specific class, what is used in classification tasks.

3.6 Unsupervised learning algorithms

Unsupervised learning algorithms work over unlabeled data. In other words, we only have access to x values of a data set. Common applications include density estimation, denoise data from some distribution or clustering similar data based on its features [5].

Principal Components Analysis is an unsupervised learning algorithm that compress data. The algorithm learns a lower dimensional representation of data than the input, and aims to make features statistically independent, removing nonlinear relationships between variables. k-means is another algorithm that divides the training set into k clusters that are near to each other.

3.7 Reinforcement learning algorithms

Reinforcement learning algorithms presupposes an agent acting in an environment and learning from experience, computing the feedback of an action through a reward function and evaluating if a certain action gives a good or a bad result [16]. Reinforcement learning uses the observed reward and searches for an optimal policy for a given environment.

Given a set of internal states $S = \{s_1, ..., s_i\}$ and a set of actions $A = \{a_1, ..., a_j\}$, at each iteration step i the agent receives a state s_i and takes an action based on a set of policies. The agent computes r_i, a scalar reward that will be received from an environment at each state transition, with the goal of maximizing its accumulated reward given a sequence of actions.

Q-Learning is a model-free technique that is used to approximate an optimal action-value function $Q(s, a)$ that measures the actions expected reward in the long-term. Approximate a Q-value function is difficult, and usually a parameterized function $Q(s, a, \Theta)$ is used to approximate $Q(s, a)$, where the agent learns Θ parameters by environment interaction.

Deep Q-Networks apply the same principle of Q-Networks, but uses nonlinear activation functions in a Convolutional Neural Network (see Section 4.3 for more information about Convolutional Neural Networks) that is able to create informative features to represent the internal states of reinforcement learning. Updating Q-values is done at each iteration and is computed according to Equation 3.8, where r is the reward, E is the expectation computed over all transition tuples of actions a in state s, λ is a discount factor for future rewards defined in range 0 to 1.

$$Q_{i+1}(s, a) = E[r + \lambda max_{a'} Q_i(s', a'|s, a)] \tag{3.8}$$

4. NEURAL NETWORKS

Machine learning algorithms are efficient to solve many types of problems, but is not efficient to deal with central problems in AI such as speech recognition and object recognition [5].

Traditional machine learning algorithms doesn't scale well with high-dimensional data. This phenomenon is called curse of dimensionality, where the number of distinct possible configurations of a set of variables increases exponentially as the number of variables increases. In practice, if the number of training examples is not big enough, the algorithm run the risk that the number of possible configurations can be bigger than the number of training examples. Under this situation, the algorithm cannot generalize well and fail to answer to questions about unseen information.

Feed-forward networks, also known as multilayer perceptron (MLP), aim to approximate functions, defining a mapping of $y = f(x; \Theta)$ by learning the Θ parameters [5]. Neural networks can be compared to functions chained together that form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$.

The advantage of using neural networks instead of traditional machine learning algorithms is that neural networks can be applied to any kind of problem by using a linear function as model. Machine learning algorithms, on the other hand, require a model with adequate capacity to be specified manually for each problem [5].

Figure 4.1 illustrate an example of a neural network as a directed graph, where each node is called an activation unit and the edges connecting activation units show how values flow through network. This model is called feed-forward network because data flow from input units x_n, passing through hidden units h_n to output unit y.

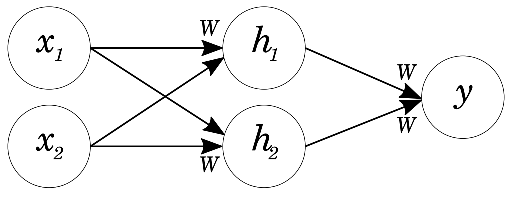


Figure 4.1 – A simple neural network example

A neural network can be separated in layers: one input layer composed of x_i activation units, m hidden layers composed of h_j activation units, and one output layer composed of y_k units. The number of activation units per layer is not related to previous or next layer.

Each activation unit inside the hidden layer and output layer receives information from all activation units from previous layer[1]. The influence of each connection is adjusted by a vector of weights $W \in \mathbb{R}^n$ associated to each activation unit, where n is the number of input connections.

[1]There are dropout techniques that reduces this number of connections between activation units.

The previous neural network can be expressed as equations, where $x \in \mathbb{R}^n$ is a vector of n entries associated to each activation unit in the input layer.

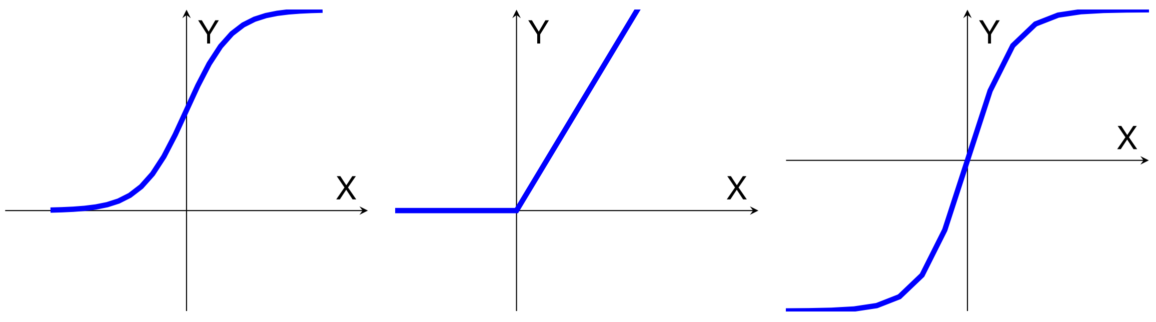
$$h_1 = g^{(1)}(W^{(1)T}x + b^{(1)}) \tag{4.1}$$

$$h_2 = g^{(2)}(W^{(2)T}x + b^{(2)}) \tag{4.2}$$

$$y = W^T h + b \tag{4.3}$$

4.1 Activation functions

Activation units compute the input values balanced with its weights, and before output its value to the next layer, apply an activation function $g(z)$ over the result. An activation function is used to transform the output of a linear function $f(x) = W^T x + b$ into a nonlinear function. Common activation functions are sigmoid function, Rectified Linear Unit (ReLU) and Hyperbolic function, shown in Figure 4.2.



(a) Sigmoid function (b) ReLU function (c) Hyperbolic tangent function

Figure 4.2 – Activation functions applied to neural network units.

Sigmoid function, defined by Equation 4.4, was widely used as activation function. It fall in disuse on traditional feed-forward networks because most of its values saturate over its domain, making difficult for optimization algorithms determine how to adjust model parameters. Sigmoid functions are still in use mainly in Recurrent Neural Networks, in many probabilistic models and some auto-encoders [5].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.4}$$

Hyperbolic tangent function, defined by Equation 4.5, shares some similarity with sigmoid function and both have similar applications, with hyperbolic tangent performing better than sigmoid function [5].

$$g(z) = tanh(x) \tag{4.5}$$

Rectified Linear Unit (RELU), defined by Equation 4.6, is the recommended default activation function used in modern neural network architectures [5].

$$g(z) = max\{0, z\} \tag{4.6}$$

Classification problems return a vector $\hat{y} \in \mathbb{R}^n$, where n is the number of classes. For each class is computed the probability (values in range 0 and 1) of an input x against n classes. Softmax functions (Equation 4.7) is responsible for generating an output vector $\hat{y}$, where i is an index for each class.

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{4.7}$$

4.2 Back Propagation

The same as Machine Learning, neural networks need to be trained so it can generalize information based on the training set. During the learning process, data flows first from input to output layer and thereafter from the output layer to input layer, where this last movement is called back propagation.

In a supervised learning approach, back propagation computes the cost function for the training set and apply stochastic gradient descent optimization procedure that consist of adjusting the parameters (weights and bias) of each activation unit.

Gradient Descent consist of computing the average cost function in a set of observed examples given input $x^{(i)}$ and $y^{(i)}$ for each i example with Θ parameters (Equation 4.8).

$$\nabla_\Theta J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\Theta L(x^{(i)}, y^{(i)}, \Theta) \tag{4.8}$$

After computing the cost at output layer, for each layer in topology we convert the gradient on layers' output into a gradient with pre-nonlinearity activation (Equation 4.9), where $a^{(k)}$ refers to an activation unit in layer k and uses element-wise multiplication.

$$g \leftarrow \nabla_a^k J = g \odot f'(a^{(k)}) \tag{4.9}$$

Next, we adjust the parameters bias and weight, in layer k, taking into account weight decay regularization method with λ weight balance, and Ω recovers original Θ parameters (Equations 4.10 and 4.11).

$$\nabla_b^{(k)} J = g + \lambda \nabla_b^{(k)} \Omega(\Theta) \tag{4.10}$$

$$\nabla_W^{(k)} J = g + h^{(k-1)T} + \lambda \nabla_W^{(k)} \Omega(\Theta) \tag{4.11}$$

Finally, gradients are propagated to lower-level hidden layer's activation.

$$g \leftarrow \nabla_h^{k-1} J = W^{(k)T} g \tag{4.12}$$

4.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a neural network specialized to compute data structured as a grid-like topology. This kind of network is commonly used in image and video processing, where data is stored into an *n*-dimensional matrix. Some common applications in computer graphics field include edge detection, but this concept can be extended, for example, to applications that require noise reduction of inputted data [5].

This architecture leverages three important ideas that can improve a machine learning system:

- Sparse interaction permits the use of variable length data instead of neural networks that have a number of input values of an unit constrained to the number of previous unit output.
- Parameter sharing refers to the use of the same parameters for more than one function or unit in the model.
- Equivariant representations guarantees that if the input data varies, the output varies in the same way.

While standard neural networks operate with matrix multiplications, convolutional networks use convolution operation, that consist of averaging a given input data against a weight function or kernel, resulting into a smoothed output (Figure 4.3).

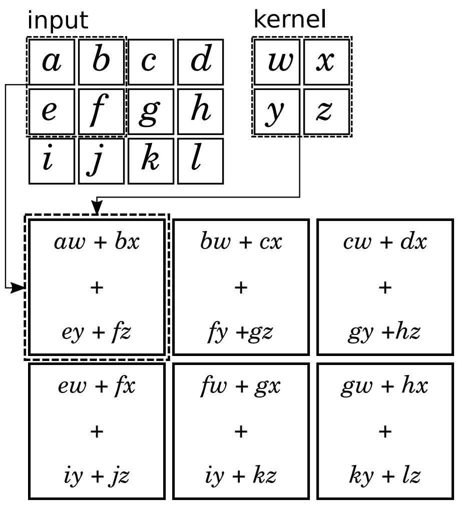


Figure 4.3 – Application of convolution operator

The kernel is a *n*-dimensional matrix that store weights for each computed cell. To compute a feature value for a coordinate location (i, j) in kth feature map of lth layer $z^l_{i,j,k}$, we apply Equation 4.13, where w^l_k and b^l_k are weight and bias values, and $x^l_{i,j}$ is the input values. The $z^l_{i,j,k}$ is used as parameters to a nonlinear activation function like sigmoid, hyperbolic tangent or ReLU.

$$z^l_{i,j,k} = {w^l_k}^T x^l_{i,j} + b^l_k \tag{4.13}$$

Pooling is an operation that return a summary statistic of the nearby outputs. For example, *max pooling* return the maximum output within a rectangular neighborhood, while *average pooling* returns the average of that neighborhood. Usually this operator produce the invariance to translation of pixels into an image (equivariant representation property).

Fully-connected layers take all neurons in previous layers and connect them to every single neuron of current layer in order to generate global semantic information.

Figure 4.4 shows the LeNet-5 network architecture that was applied in handwriting recognition tasks. It receives as input a 32x32 matrix and apply a series of convolution operations and pooling (subsampling layer) operations. At final stages, a fully-connected layer is applied to generate a final result.

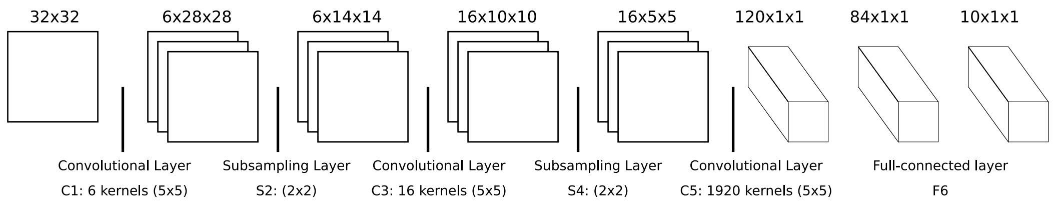


Figure 4.4 – LeNet-5 CNN architecture

Convolutional Neural Networks are gaining space in text processing tasks like text classification [6]. Chopra et al., in their summarization technique, used CNN in his summarization method to encode input text [2].

4.4 Recurrent Neural Network

Recurrent Neural Network (RNN) is a specialized type of neural network for sequential data processing. The key differences of Recurrent Neural Network compared to feed-forward networks is the parameters sharing across different parts of the model, where the network uses the same weight values for all activation units to process input data at different time steps [5].

Figure 4.5 show a Recurrent Neural Network where, for each time step t, $h^{(t)}$ is a hidden unit that receive input from $x^{(t)}$, σ is an output unit that feed a cost function unit L together with y provided by a labeled training set. U, V and W are weights associated respectively to input $x^{(t)}$, input $\sigma^{(t)}$ and input h^t.

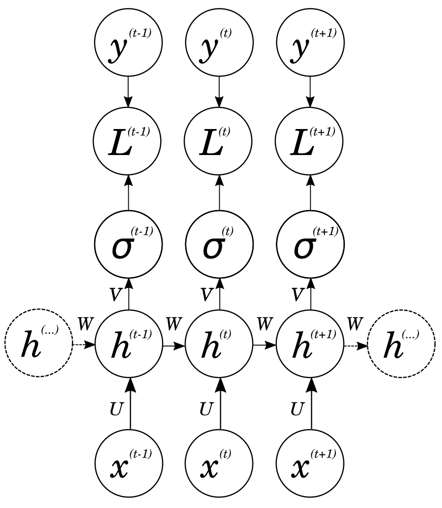


Figure 4.5 – A recurrent neural network with output units at each time step

Figure 4.6 receives $x^{(t)}$ values at each t time step but, instead of producing an output $y^{(t)}$ value at each time step, computes the loss function L for the whole time and generate a single y output value. Its activation units can be expressed by the following equations:

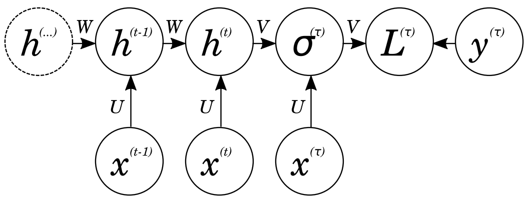


Figure 4.6 – A simple recurrent neural network with a single output unit.

$$h^{(t)} = tanh(b + Wh^{(t-1)} + Ux^{(t)}) \tag{4.14}$$

$$\sigma^{(t)} = c + Vh^{(t)} \tag{4.15}$$

$$\hat{y} = softmax(\sigma^{(t)}) \tag{4.16}$$

Bidirectional Recurrent Neural Network (Figure 4.7) allows a model to learn not only from previous data but from forthcoming data as well. Common applications include speech recognition, handwriting recognition and bioinformatics [5].

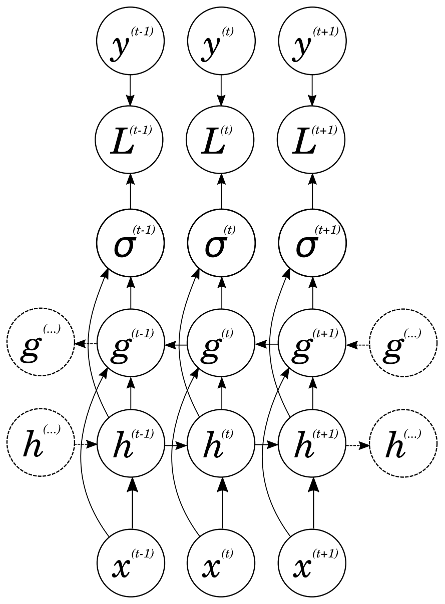


Figure 4.7 – A simple bidirectional recurrent neural network.

This network has two hidden units: $h^{(t)}$ that run forward, and $g^{(t)}$ that run backward. Both hidden units receive input from $x^{(t)}$ and $g^{(t)}$ also receives input from $h^{(t)}$ to compute the output $\sigma^{(t)}$.

4.4.1 Long-Term Dependencies

The key difference of Recurrent Neural Networks is reusing previous computed information. Under some situations is important to remember specific information seen long before. During gradient propagation, however, information tend to either vanish or explode [5].

Proposed workarounds include adding skip connections from one time step to another further in the network to more complex solutions like using specialized unit types, that result in performance improvement of tasks like speech recognition, hand writting recognition and generation and machine translation.

Long Short-Term Memory (LSTM) is a type of activation unit that uses a self-loop structure to produce paths where gradient can flow for long duration. This unit have four input channels: two responsible for input values and two that control how the unit behave (Figure 4.8).

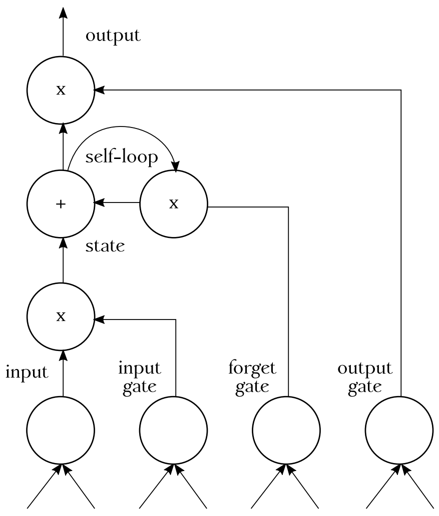


Figure 4.8 – A Long Short-Term Memory unit.

The input receives values from other hidden units. The forget gate (Equation 4.17) defines if the retained information must be disposed, where $b^{(t)}$ is bias value assigned to forget gate, $x^{(t)}$ is the current input, U^f and W^f are respectively weights for input and forget gate.

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}) \tag{4.17}$$

The internal state of LSTM is updated by $s^{(t)}$ (Equation 4.18), taking into account a conditional self-loop weight $f_i^{(t)}$, where $g_i^{(t)}$ denotes the input gate, b^t, U^t and W^t denote respectively biases, input weights and recurrent weights.

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} z_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}) \tag{4.18}$$

The external input gate $g^{(t)}$ (Equation 4.19) is computed in similar way as forget gate, but with its own parameters.

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{t-1}) \tag{4.19}$$

The output value $h_i^{(t)}$ (Equation 4.20) computes the output value of this unit, taking into account the weight computed by output gate $q_i^{(t)}$.

$$h_i^{(t)} = tanh(s_i^{(t)}) q_i^{(t)} \tag{4.20}$$

The output gate $q_i^{(t)}$ allow the output $h_i^{(t)}$ to be shut off (Equation 4.21), where W_i are weight parameters that are learned during training process.

$$q_i^{(t)} = \sigma(b_i^0 + \sum_j U_{i,j}^0 x_i^{(t)} + \sum_j W_i^{(0)} h_j^{(t-1)}) \tag{4.21}$$

Gated Recurrent Units (GRU) work in similar manner than LSTM, with main difference that a single unit controls forget gate and update gate of the state unit. The output value $h^{(t)}$ is computed according to Equation 4.22, where $u^{(t)}$ stands for update gate (Equation 4.23) and $r^{(t)}$ for reset gate (Equation 4.24). U^u and W^u are weights for input x and output $h^{(t)}$ applied to update gate, and U^r and W^r are the same but applied to reset gate.

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)})\sigma(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}) \tag{4.22}$$

$$u_i^{(t)} = \sigma(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}) \tag{4.23}$$

$$r_i^{(t)} = \sigma(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r hj^{(t)}) \tag{4.24}$$

5. NEURAL NETWORKS AND TEXT PROCESSING

5.1 Word vector representation

As introduced in Section 2.2, word vectors are derived from machine learning processes, which require substantial training data to generate a meaningful representation of words and their relationships [11].

The training process over a data set generates a model and the word vector representation. The RNN architecture applied to generate word vectors is shown in Figure 5.1.

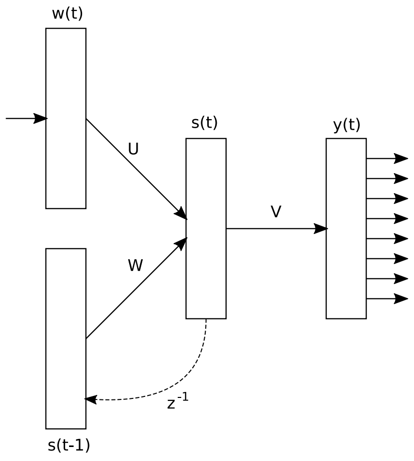


Figure 5.1 – A recurrent neural network applied to generate word vectors.

where $s(t)$ (Equation 5.1) is an hidden layer that retains the sequence history, $w(t)$ represent input word at time t as an one-hot vector, $y(t)$ (Equation 5.2) is the probability distribution over words, where $w(t)$ and $y(t)$ have the same length. The columns of U vector contains the representation of words after the training process.

$$s(t) = f(Uw(t) + Ws(t-1)) \tag{5.1}$$

$$y(t) = g(Vs(t)) \tag{5.2}$$

$$f(z) = \frac{1}{1+e^{-z}}, g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \tag{5.3}$$

Figure 5.2 shows word vectors distributed into a 2-D space. As we can see words are grouped based on shared features: royalty, countries, animals, materials and literary context.

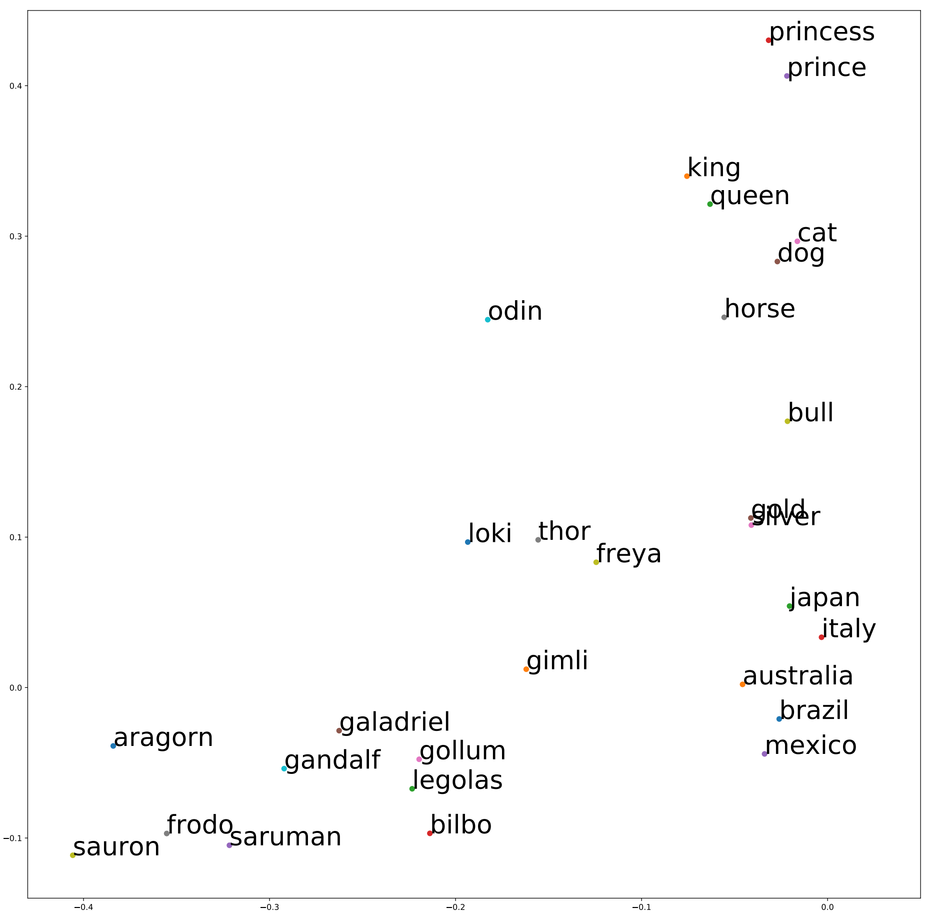


Figure 5.2 – Word vectors visualization over a 2-D space.

5.2 Sentence vector representation

Vector representation can also be used to capture sentence features and relationship between its words. Pagliardini et al. [14] proposed an unsupervised learning model named *Sent2Vec* that converts sentences into a vector representation.

Sent2Vec uses the same principle as converting words to vectors, but augmenting it by learning vectors not only to words but also to n-grams present in each sentence. This process is accomplished by averaging sentences word vectors, as shown in Equation 5.4, where $R(s)$ is a list of n-grams present in sentence S.

$$u_s = \frac{1}{|R(s)|} \sum_{w \in R(S)} u_w \tag{5.4}$$

To illustrate how sentence vectors capture the meaning of sentences, we recovered our previous examples in Table 2.1 used to illustrate ROUGE scores, and computed similarity based on distance between its vectors, as shown in Table 5.1.

Sentence 1	Sentence 2	Distance
exact sentences get full scores	exact sentences get full scores	0.0
word	word	0.0
one simple test	another simple test	0.01
sentence one	Sentence one	2.30
summarization returns a short version of content	summarization techniques generate a shorter interpretation of text	0.62
this is a random sentence	something else	1.92

Table 5.1 – Examples of sentences similarity based on distance between vectors.

Observing the results, sentence vectors capture semantic similarities between sentences, taking into account our examples with exact match (*exact sentences get full scores*), sentences with semantic similarity like *one simple test* and *another simple test*, *summarization returns a short version of content* and *summarization techniques generate a shorter interpretation of text* that presents low distance values. Our test case *sentence one* and *Sentence one* presents high distance value because our dictionary only contains lower case words, where upper case words return a vector of zero values.

5.3 Encoder-decoder sequence-to-sequence architectures

The encoder-decoder architecture allows a Recurrent Neural Network to receive as input a sequence of values of *n* length and return another sequence which is not necessarily of the same length. Applications as speech recognition, question answering systems and automatic text summarization benefit from this structure [5].

The input of an RNN are often called "context". The goal is to produce a representation of this context C, that might be a vector or a sequence of vectors that summarize the input sequence $X = (x^{(1)}, ..., x^{(n_x)})$. The main idea is that an encoder process the input sequence and produces the context C. After, a decoder conditioned on that fixed-length vector is used to output a sequence $Y = (y^{(1)}, ..., y^{(n_y)})$ where n_x and n_y can be distinct values. Traditional architectures constrained input x and output y to have the same length. An example is shown in Figure 5.3.

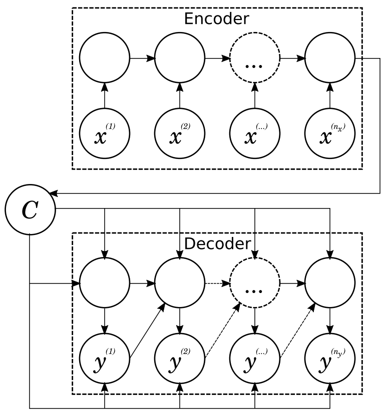


Figure 5.3 – A encoder-decoder architecture

6. IMPLEMENTATION AND DATA SET

6.1 Data set

The data set is composed of news articles extracted from CNN and DailyMail websites that was provided by Nallapati et al. [13]. This data set was generated based on an earlier data set curated by Hermann et al. [7] and applied on tasks like question answering. Both data sets are structured by a headline, a story and `@highlight` sentences that enumerate its key ideas. A sample of this data set is shown in Figure 6.1.

Both data sets contain stories about politics, economics, sports, entertainment and technology, to name a few classes. Table 6.1 summarizes general statistical features about this data set.

		Sentences		Words	
	Stories	Total	Average	Total	Average
CNN	92,579	3,103,041	34	75,259,364	813
DailyMail	219,506	6,032,961	27	182.750.863	833
Total	**312,085**	**9,136,002**	**30.5**	**258,010,227**	**822.5**

Table 6.1 – Data set statistical features.

6.2 Data set Pre-processing

Pre-processing a data set consist of preparing its raw content in a way that it can be used in our application. We break the text into sentences and extract the ground-truth summary that will be used for further evaluation. For ground-truth summary, we selected the sentences that follows the `@highlight` tag. In our example shown in Figure 6.1, the summary is *"turkey's ruling party agrees to lift ban on head scarves in universities. ban introduced after military coup in 1980 as seen as a sign of religion. murkey is a secular nation but its population is mainly muslim. proposal has brought protests among the secular population"*.

We observed in preliminary experiments that data set pre-processing have a big impact over final results, where precision metrics doubled its scores after fine tuning this process. Because of this, we defined some constraints like discarding texts shorter than 10 sentences and longer than 45 sentences, discarding sentences shorter than 30 characters, and converting all characters to lower case because our dictionaries contain only lower case words.

(CNN) – Two of Turkey's main political parties are pushing for a constitutional amendment to lift bans on headscarves at public universities, a move that has caused concern among Turkey's secular population. The lifting of the ban on headscarves has caused concern among Turkey's secular population.

Prime Minister Recep Tayyip Erdogan initiated the move, saying it would create equality in Turkey's higher education.

The constitutional commission will discuss the proposal – submitted by the AKP and MHP parties – in the coming days before sending it to the floor for a vote.

If approved, it would need President Abdullah Gul's approval, which is expected.

Under the proposal, veils, burqas or chaddors – all of which cover a woman's face – would not be allowed.

Bans on headcoverings were imposed in the early 1980s by Turkey's universities because they were seen as political symbols and conflicted with Turkey's secular governing system.

The proposal to change Turkey's constitution sent chills through Turkey's secular population. Women's groups went to parliament Tuesday to voice their rejection.

"This is a direct threat to the republic and its foundations," said Deniz Baykal, leader of Turkey's main secular party, CHP.

Another CHP lawmaker said she fears that if the proposal is enacted, parents will feel pressure to have their daughters wear headscarves, even in elementary school.

Mustafa Akaydin, head of Turkey's Higher Education Commission, is against the proposal. He said that allowing headscarves would be a rejection of Turkey's secular system of government.

"It is an attempt to create a counterrevolution," Akaydin said. "It will be a breaking point."

He said a majority of female high school students at one school were wearing headscarves during last weekend's entry exams – a rarity in Turkish schools.

The Higher Education Commission will meet Friday in Ankara to discuss the proposed changes. E-mail to a friend

@highlight
Turkey's ruling party agrees to lift ban on head scarves in universities
@highlight
Ban introduced after military coup in 1980 as seen as a sign of religion
@highlight
Turkey is a secular nation but its population is mainly Muslim
@highlight
Proposal has brought protests among the secular population

Figure 6.1 – Sample extracted from CNN stories data set

6.3 Word vectors processing

We used two word vectors dictionaries in our experiments:

- One pre-trained dictionary[1] trained over English Wikipedia dumps, containing a total of 1,066,988 words, where each word is represented by a vector of 600 floating point values.
- A dictionary trained by us over Wikipedia dumps with 2,809,163 words where each word is represented by a vector of 100 floating point values.

6.4 Implementation tools

We implemented our models using Python as programming language because of available libraries and frameworks that allow us to do fast prototyping and experiments, using the following support libraries:

- Numpy library for numerical processing and data storage.
- Keras for Neural Network modeling with Tensorflow as backend, so we can benefit of Graphical Processing Unit (GPU) acceleration features.
- Natural Language Toolkit (NLTK)[2] for text parsing and pre-processing.
- *sent2vec* library[3] to train a dictionary of word vectors based on text and also to convert sentences into vectors.
- ROUGE library[4] to compute text summarization evaluation metrics.

Our code is available at GIT repository https://github.com/mauriciosteinert/pucrs-cc-tcc-2018/.

[1]Word vectors dictionary `sent2vec_wiki_unigrams` available at https://github.com/epfml/sent2vec
[2]https://www.nltk.org/
[3]https://github.com/epfml/sent2vec
[4]https://github.com/pltrdy/rouge

7. FEED-FORWARD NEURAL NETWORK SUMMARIZATION

This Chapter describes our experiments of training a classification Feed-Forward Neural Networks for the summarization task.

In our input data all sentences in text are converted to its vector representation and appended one after the other. To have a fixed input size, we take the text with greater number of sentences and use this number multiplied by the word vector dictionary dimensionality as input size. Texts with shorter number of sentences receive padding of zero values to fill the input value. For ground-truth values, we use best ROUGE scores as metric and generate, for each example, an one-hot vector $\vec{y} = \{0, 0, 1, 0, ..., 0\}$ that consist of a vector of zeroed values with same length as our maximum number of sentences, where all of its entries have zero values except for one that represent the position of our sentence selected as summary, that is set to 1.

We test different neural network architectures to understand training process behavior and to identify which one performs better. All tests are executed over the same data set and pre-processing conditions described in Section 6.2: 20,000 training examples, 20,000 test examples, training batch size of 200 examples for 100 epochs. For all experiments in this Chapter, we use ReLU as activation function for activation units and softmax as activation function for our output unit.

7.1 Shallow Neural Networks

Shallow Neural Networks architecture is composed of an input layer, only one hidden layer, and an output layer [5]. Figure 7.1 illustrate this architecture. All of our experiments share the same features presented earlier, except for the last two setup that use dropout units between hidden layer and output layer to discard some of connections between units, reducing the amount of information that flows from one layer to another and consequently reducing our model complexity.

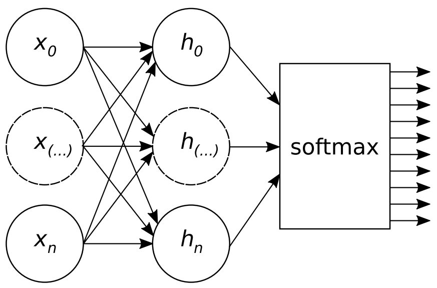


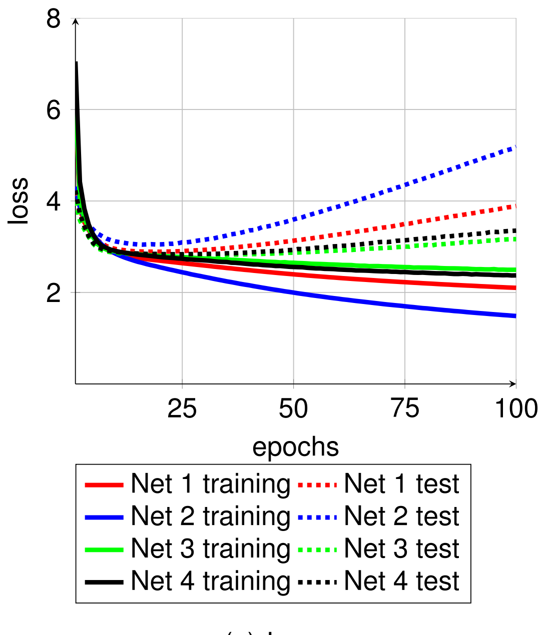
Figure 7.1 – Shallow Neural Network architecture.

We evaluate varying the quantity of activation units in hidden layers and use dropout in some situations, resulting in the following architectures:

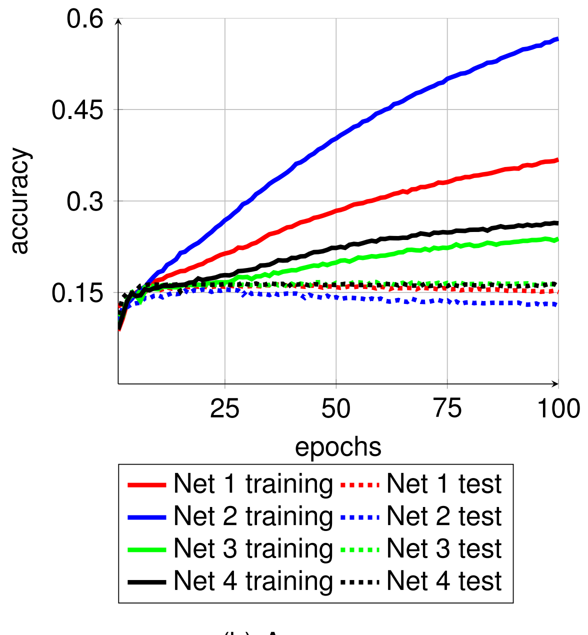
- Net 1: 8 activation units into a single hidden layer.
- Net 2: 16 activation units into a single hidden layer.
- Net 3: 8 activation units into a single hidden layer and dropout between hidden layer and output layer.
- Net 4: 16 activation units into a single hidden layer and dropout between hidden layer and output layer.

Evaluating the training process for each network, as shown in Figure 7.2, we verify that:

- All of our proposed architectures fall into overfitting condition, where training loss and test loss values diverge during training process.
- *Net 2* presents the biggest gap between training loss and test loss, possibly because this model have higher model complexity (see Section 3.4 for more details) due to high number of activation units in hidden layer. Observing accuracy scores, this architecture achieve almost 60 % accuracy for training set but stays within 20 % for test set, indicating that the architecture is memorizing results and fails to generalize the learning process over unseen data.
- Applying dropout to *Net 3* and *Net 4* reduces the gap between training loss and test loss, also reflecting into training accuracy and test accuracy. *Net 3* presents the smaller gap, but training accuracy stays within 30 % and test accuracy within 16 % after 100 epochs, not displaying signs of further improvement even with larger number of epochs. Even with more epochs, training loss and test loss still diverge at a smaller rate.



(a) Loss



(b) Accuracy

Figure 7.2 – Shallow Neural Networks loss and accuracy in function of epochs.

After analyzing loss and accuracy during training process, we conclude that all our Shallow Neural Network architectures fall into overfitting condition, failing to generalize information and consequently producing low accuracy scores when running over test set.

Our preliminary tests was done using 50,000 training examples and 64 activation units in hidden layer, what resulted in overfitting condition. Given that recommendation to workaround overfitting condition is reducing models complexity or using a large training data set [5], we reduced our models complexity and, even with 20,000 and 50,000 training examples, results are similar.

7.2 Deep Neural Networks

Deep Neural Networks have the same structure as Shallow Neural Networks, but allow a large number of hidden layers. Figure 7.3 shows our architecture.

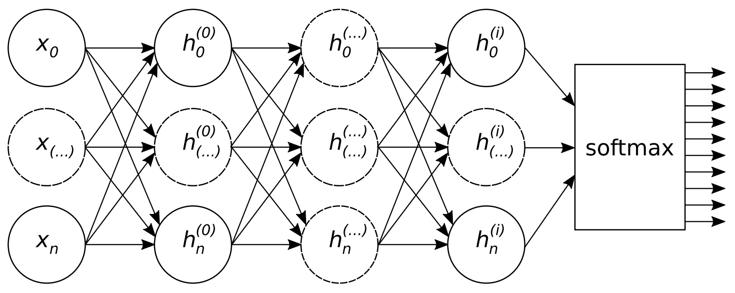


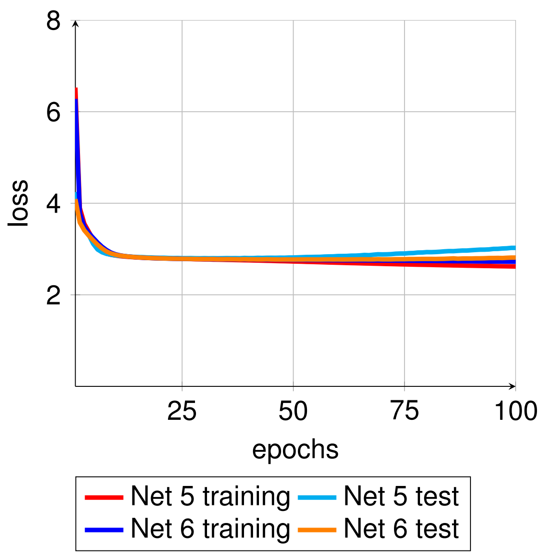
Figure 7.3 – Deep Neural Network architecture.

In this experiment, we vary the number of hidden layers as follow:

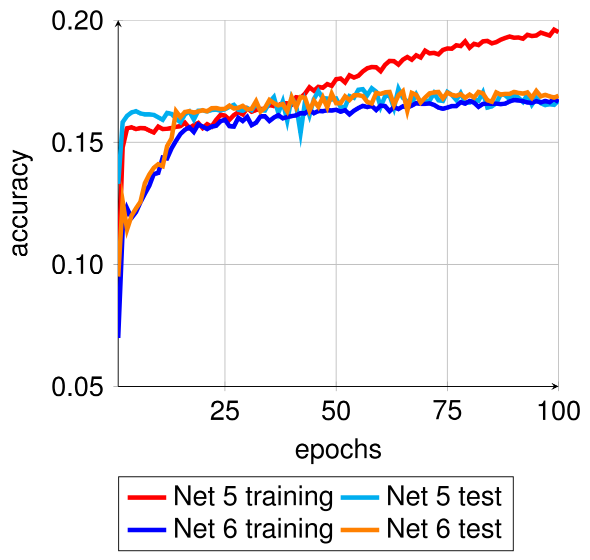
- Net 5: 2 hidden layers each with 4 activation units.
- Net 6: 4 hidden layers each with 2 activation units.

Evaluating loss and accuracy in function of epochs during training process (Figure 7.4), we verified that:

- Training loss and test loss for both *Net 5* and *Net 6* present closer values, decreasing fast in first epochs and slowing down after epoch 10.
- *Net 5* test loss starts to diverge around epoch 50, possibly because of model complexity.
- *Net 5* attain better training accuracy than *Net 6*, but test accuracy get stuck within 17 % without further improvement since epoch 50.



(a) Loss



(b) Accuracy

Figure 7.4 – Deep Neural Networks loss and accuracy in function of epochs.

Using Deep Neural Networks reduce overfitting condition, but both training accuracy and test accuracy do not improve significantly in function of epochs.

7.3 Convolutional Neural Network

The number of input units in our neural networks depend of the number of sentences we choose as maximum number of sentences for each example multiplied by dimensionality of word vector dictionary. During our preliminary experiments we did not constrained the maximum number of sentences per text and, observing that the mean of sentences per example is around 30, and the text with larger number of sentences have 106 sentences, a large number of examples receives a lot of padding.

Based on that observations, we decided to limit the maximum number of sentences per example, as detailed in Section 6.2. In the following experiments, we propose using convolution layers to reduce dimensionality of input layer. Figure 7.5 shows our proposed architecture. The convolutional layer returns a total of 16 filters, passing its results to a max pooling layer of size 3, that pass its results to a fully-connected neural network with 2 hidden layers with 8 activation units per layer using ReLU as activation function. Finally, our output layer uses a softmax activation function. We refer to this experiment as *Net 7*.

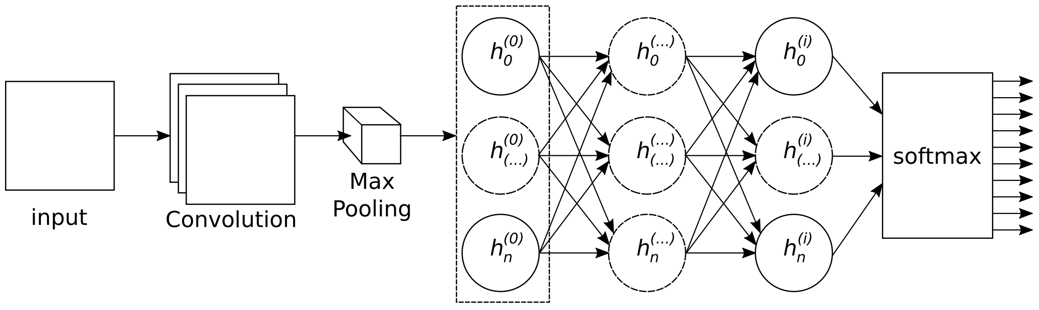
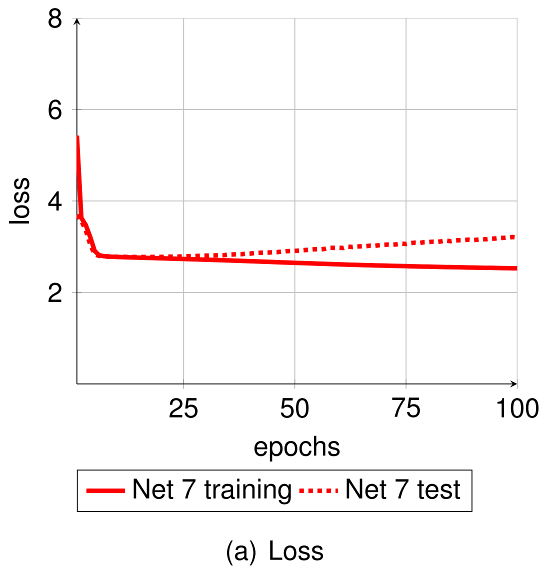


Figure 7.5 – Convolutional Neural Network architecture.

Observing loss and accuracy metrics in Figure 7.6:

- *Net 7* presents similar behavior to Deep Neural Network experiments *Net 5* and *Net 6*, except that in *Net 7* training loss and test loss diverges after epoch 35, possibly due to larger number of hidden units per hidden layer.
- *Net 7* presents high training accuracy than Deep Neural Network experiments, probably because of larger number of hidden units per hidden layer. Test accuracy, however, gets within 17 % after epoch 25.



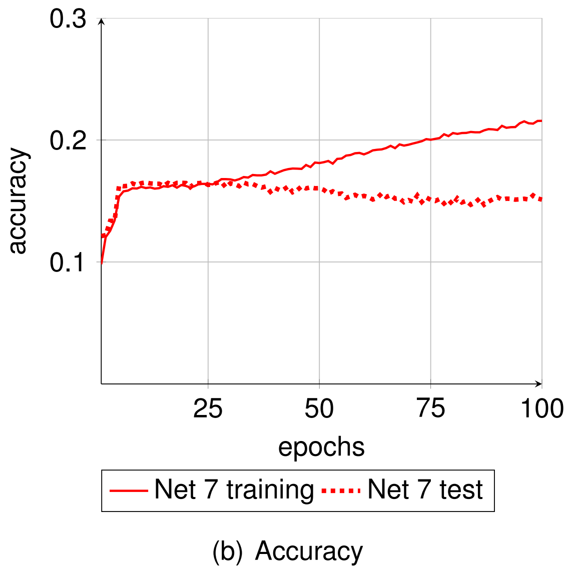


Figure 7.6 – Convolutional Neural Networks loss and accuracy in function of epochs.

Given our observed results and comparing to Deep Neural Network results, we conclude that applying convolution operators to input layer do not affect significantly the learning process. This architecture fails to generalize information and to predict over unseen data.

We also evaluated ROUGE scores for this architecture, as shown in Figure 7.7, where all experiments results are within 12 % for all F-scores. Important to note that reporting ROUGE scores for this experiment may not be adequate because we setup our experiments as a classification problem that only knows which sentence must be selected as summary, given our labels are represented by an one-hot vector. The learning process can only report exact matches, being unable to evaluate how close each sentence are from ground-truth summary.

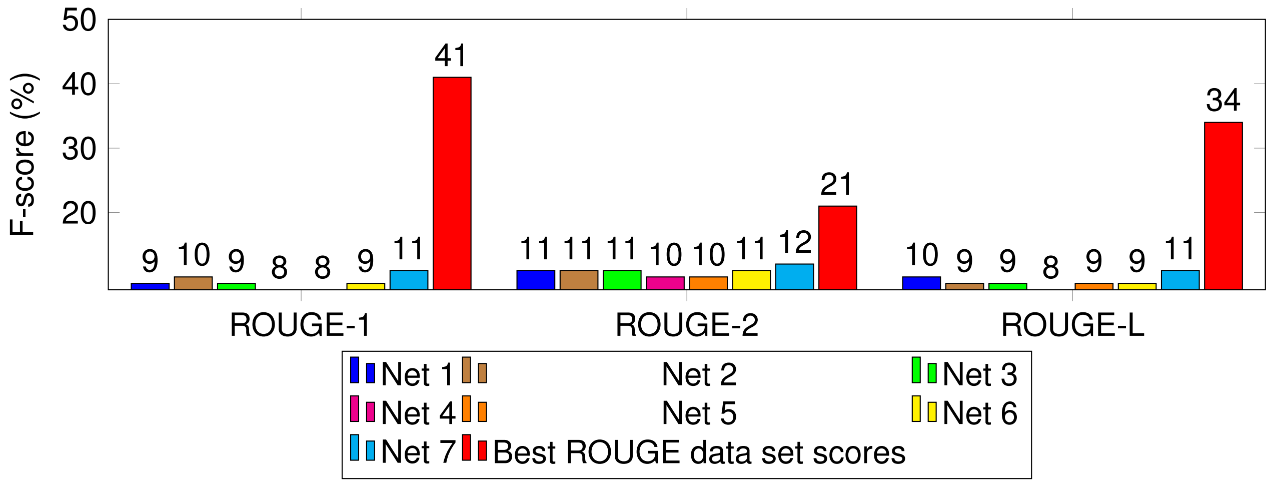


Figure 7.7 – Performance evaluation of Feed-Forward Neural Network technique using ROUGE scores.

During our experiments, we identified that our architectures return as summary index of sentences that do not exist in a given example (for example the summary of a text with 5 sentences is sentence with index 10). This phenomenon occurs because of variable text length and the neural network requisite of fixed input size, what we contour by padding shorter texts. Figure 7.8 reports the number of occurrences for each of our proposed networks. Even occurring rarely, this situation represents a serious consistency problem.

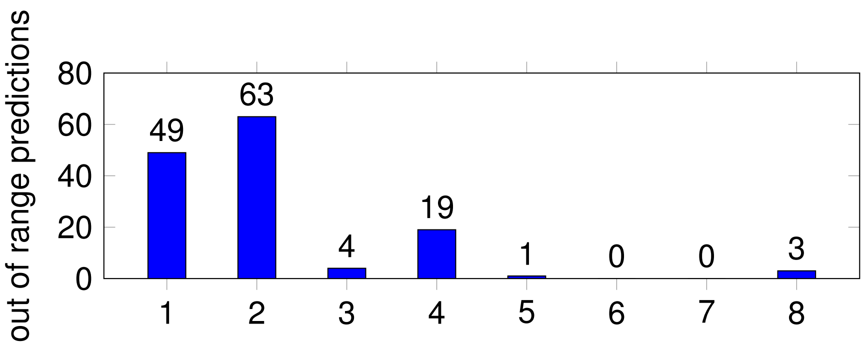


Figure 7.8 – Out of range predictions over a total of 20,000 examples.

After evaluating all set of experiments in this Chapter, we conclude that Feed-Forward Neural Networks do not present a satisfactory performance in text summarization task. The biggest problem found in all experiments are overfitting, where all workarounds applied to solve this issue do not give us the expected results. The majority of these models report out of range predictions, what compromise the reliability of this solution.

8. RECURRENT NEURAL NETWORK SUMMARIZATION

In this experiment, we implemented a supervised learning algorithm that aims to predict ROUGE scores for each sentence in texts given as input its word vector representation.

The architecture consist of a Recurrent Neural Network with Long Short-Term Memory (see Section 4.4.1) units into hidden layer to retain context information between time steps. To improve performance, we use a bidirectional network (see Section 4.4 for more details). Figure 8.1 summarizes our architecture.

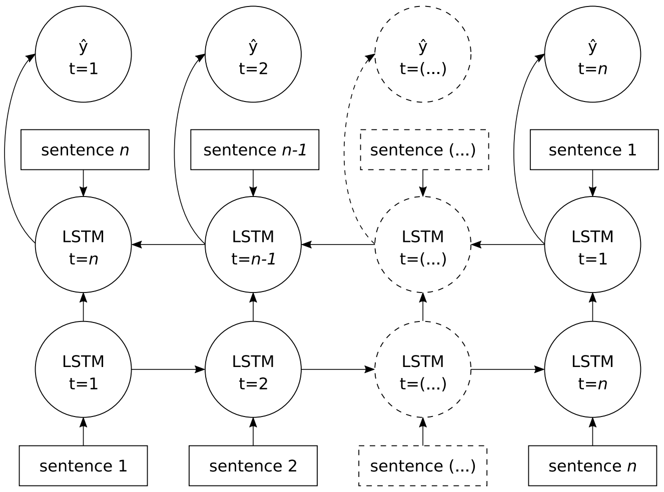


Figure 8.1 – RNN experiment architecture.

Input data consist of a matrix $X^{(n,t,f)}$, where n is the number of examples, t is the number of time steps, with each time step being a sentence, and f is the number of features for each time step, that have the same dimensionality of our word vector dictionary for each sentence. The $Y^{(n,t,f)}$ matrix contain labels for each time step, where for each sentence t we have 3 features f that are ROUGE-1, ROUGE-2 and ROUGE-L F-scores. For each computed time step, our model return a vector $\hat{y}$ with predicted ROUGE F-scores for the current sentence. Figure 8.2 illustrate the data set design matrices X and Y used to feed our recurrent network.

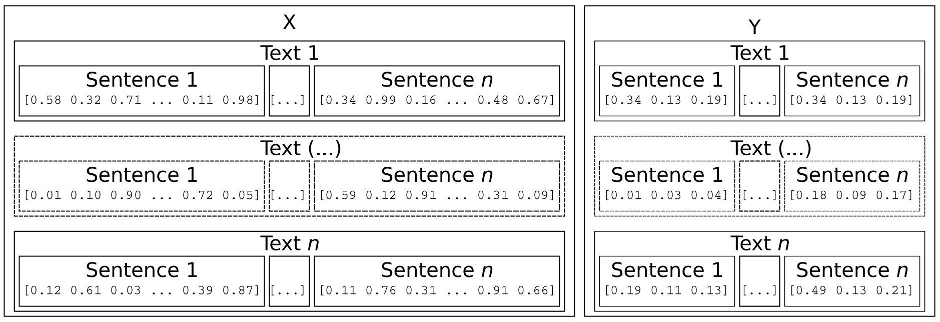


Figure 8.2 – RNN experiment data set structure.

We trained our model *RNN 1* over a data set of 20,000 examples using a word vector dictionary of dimensionality equal 100 (see Section 6.3 for more details), training batch size of 2,000 examples for a total of 100 epochs. We also trained a second model named *RNN 2* using the same conditions previously stated but using a word vector dictionary of dimensionality equal 600.

To evaluate our results, we used a test data set with 20,000 examples. Data set pre-processing are according to previously established constraints discussed in Section 6.2. Taking into account that our neural network only accept input values of fixed size, we applied padding with zero values to texts shorter than 45 sentences until all of them had the same size.

The architecture is composed of 8 LSTM hidden units that uses ReLU as activation function. Mean Squared Error is used as loss function and Adam optimizer to define how the parameters must be updated at each back-propagation pass.

Figure 8.3 shows how loss and accuracy values evolve in function of epochs. Evaluating our results:

- *RNN 1* trainig loss and test loss values present fast decreasing in first epochs, with a monotonic behavior for both values. Training accuracy and test accuracy increases in function of epochs, but present accentuated fluctuations, specially over the test set. Training accuracy keeps improving with small fluctuations after each epoch.
- *RNN 2* training loss and test loss present a fast decreasing in first epochs, and test loss values diverges abruptly around epoch 40. Training accuracy increases fast during first epochs, decreases within epoch 25 and fluctuate below 70 %, while test accuracy presents accentuated fluctuations. Given that *RNN 2* uses a dictionary with half number of words, this may impact performance accuracy because of out of vocabulary words.

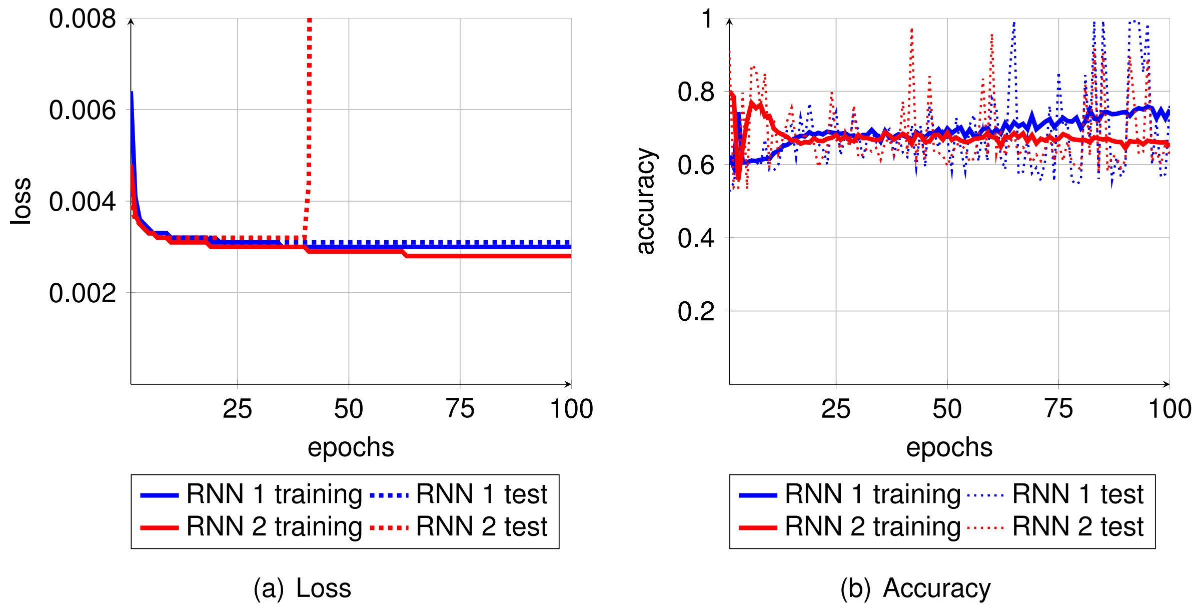


Figure 8.3 – RNN loss and accuracy in function of epochs.

ROUGE F-scores are reported in Figure 8.4, where we select as predicted summary the sentence in $\hat{Y}$ with highest ROUGE F-score. Important to note that *RNN 2* does not present performance improvements by using a larger word vector dictionary when compared to *RNN 1*. Observing the error bars, we observe that standard deviation does not vary significantly when comparing our results with best ROUGE F-scores attained over this data set.

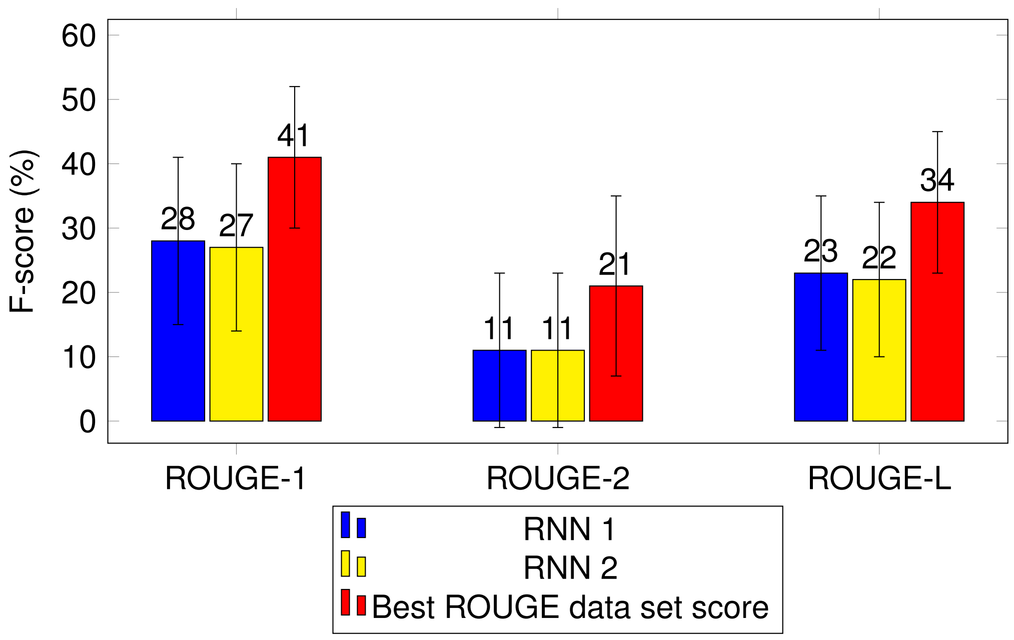


Figure 8.4 – Performance evaluation of RNN technique using ROUGE scores.

In *RNN 2* test case, giving the abrupt increasing of test loss around epoch 40, we trained our network using early stopping around epoch 30, where we report exactly the same ROUGE scores obtained after 100 epochs.

We also checked for out of range predictions, that is where predicted sentence index is higher than the total of sentences available in text, what can happen because of our fixed input data size and the existence of texts shorter than this value. This method does not predict any out of range sentence given our test data set.

9. VECTOR OFFSET SUMMARIZATION

The vector offset summarization method is inspired by Aires et al [1], which uses a similar approach to evaluate norm conflicts in contracts. Our extractive summarization method consist of an unsupervised learning approach that identify which sentence in a text best represents its overall content.

We use word vectors feature of capturing similarities between words and sentences, what allow us to operate over vectors and perform analogies like the example $\vec{x}_{king} - \vec{x}_{man} + \vec{x}_{woman} \approx \vec{x}_{queen}$ (see Section 5.1). Instead of operating over word vectors, in this case we operate over sentence vectors.

Our process is divided into the following steps (Figure 9.1):

1. Data set pre-processing, as detailed in Section 6.2. We evaluate this method using the word vector dictionaries discussed in Section 6.3.
2. Compute $\vec{mean}$ of whole text (Equation 9.1), where $\vec{s}_i$ represent each sentence vector in text and m represent total of sentences.
3. Compute $\vec{offset}$ by subtracting the ground-truth vector $\vec{y}$ from $\vec{mean}$ (Equation 9.2).
4. Compute the distance between the mean of whole text and each candidate sentence plus $\vec{offset}$ previously computed, choosing as summary the sentence that is closer to whole text vector (Equation 9.3)

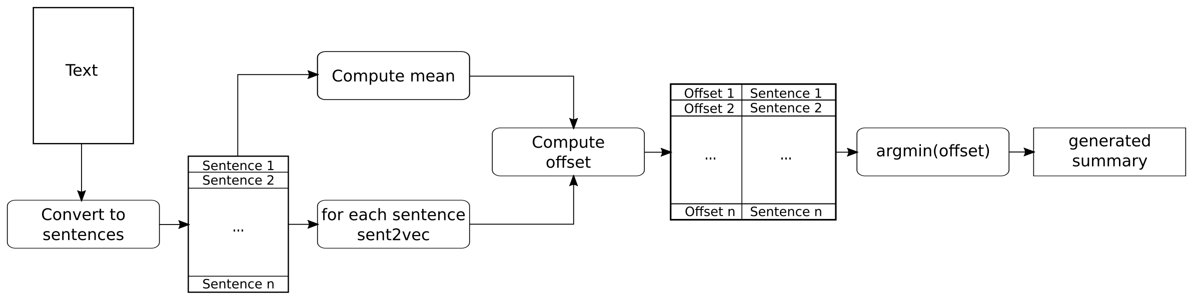


Figure 9.1 – Vector offset summarization workflow.

$$\vec{mean}_{text} = \frac{1}{m}\sum_{i}^{m} \vec{s}_i \tag{9.1}$$

$$\vec{offset} = \vec{mean}_{text} - \vec{y} \tag{9.2}$$

$$summary = argmin(\sum_{n=0}^{m}\sum_{i,j} |abs(\vec{offset}_{i,j} + \vec{s}_{n(i,j)})^2|^{\frac{1}{2}}) \tag{9.3}$$

After processing 298,017 texts using word vector dictionaries of dimensionality 100 and 600 (Figure 9.2), we obtained F-scores in our best case of 33 % for ROUGE-1, 14 % for ROUGE-2 and 28 % for ROUGE-L.

Using a word vector dictionary with higher dimensionality indeed results in better performance when compared to using a dictionary with lower dimensionality. Performance improvement is not proportional to vector length, possibly because our vector of dimensionality equal 600 possess half the number of words than our dictionary with dimensionality 100, what possibly result in a high number of out of vocabulary occurrences. Higher dimensional dictionaries, however, require more memory and are more computationally expensive to train and calculate. A smaller dictionary (100 floating points for each word), that requires 6 times less computational resources results in a precision reduction of 3 % when compared to our bigger dictionary. The best approach of what to use must be evaluated given the requirements of our problem: if precision is key and we have plenty of computational resources, using a higher dimensional dictionary is the best choice. If we are low of computational resources and response time is more important than precision, a dictionary with less dimensions can still do a good job.

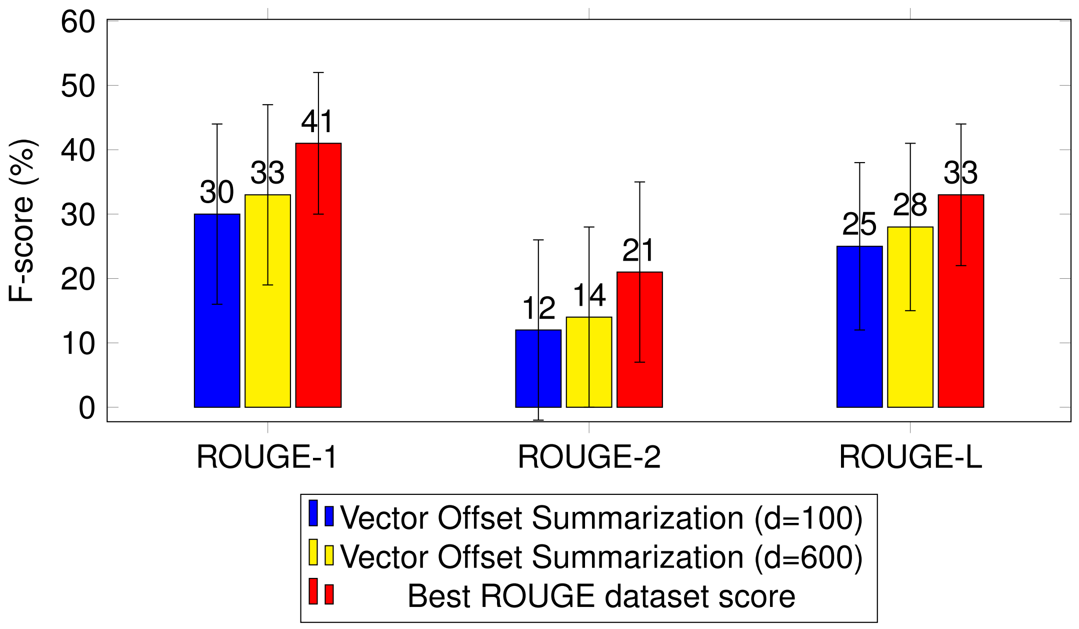


Figure 9.2 – Performance evaluation of vector offset technique using ROUGE scores.

Analyzing standard deviation values through error bars, we observed that this metric does not change drastically when comparing best ROUGE scores to both of our experiments.

To make a qualitative analysis of achieved results, we computed in how many examples our method select as summary sentences that are in top 10 of best ROUGE scores, where we verified that, in the best case, in 35 % of examples the best sentence is selected as summary and in approximately 80 % of examples the selected sentence is within top 6 best ROUGE scores (Figure 9.3).

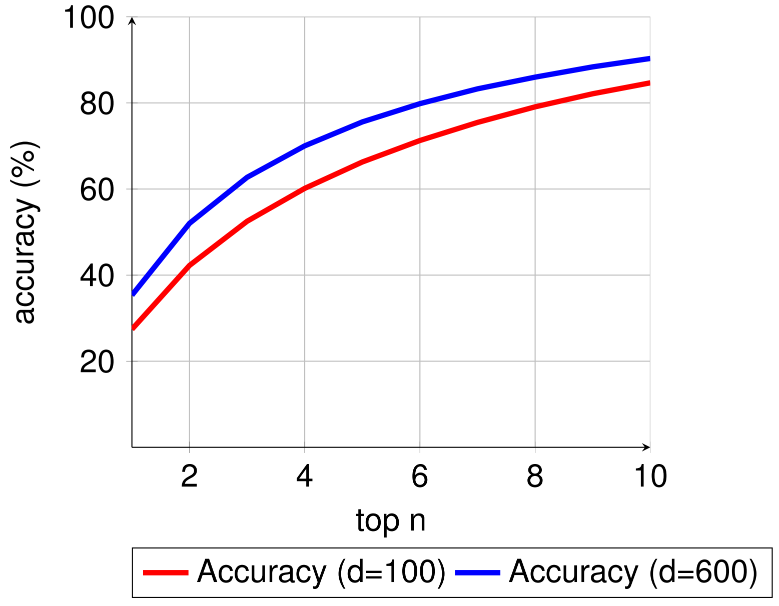


Figure 9.3 – Summarization accuracy given top 10 best ROUGE scores.

Given the observed results, this method is simple to implement, does not require previous training and have low computational requirements, but yields satisfactory and reliable results when compared to best ROUGE scores over this data set.

10. QUALITATIVE ANALYSIS

In this Chapter we manually evaluate some results of our summarization methods from a readers point of view, taking into account aspects like conciseness and completeness. We evaluate results generated by Recurrent Neural Network and Vector Offset summarization methods, comparing them with ground-truth values and what is best sentence considering ROUGE scores.

Example in Table 10.1 is about an incident with guns in a school giving as culprit the north American Second Amendment that allow weapons possession and how this affect border countries like Mexico. Extracted ground-truth captures the key idea in text and, for this example, all of our summarization methods choose the best sentence based on ROUGE scores as summary.

		ROUGE		
Method	Summary	1	2	L
Ground-truth	Mexico blames brutal drug violence on guns trafficked from the United States. Ambassador: The Second Amendment was not intended to arm foreign criminals. He says indignation after the Connecticut school shooting may open "a window of opportunity"	-	-	-
RNN	the country's top diplomat in the united states says the tragic connecticut school shooting may have "opened a window of opportunity" for obama to fix a problem that has long plagued both sides of the border.	34 %	16 %	32 %
Vector Offset	the country's top diplomat in the united states says the tragic connecticut school shooting may have "opened a window of opportunity" for obama to fix a problem that has long plagued both sides of the border.	34 %	16 %	32 %
Best Sentence	the country's top diplomat in the united states says the tragic connecticut school shooting may have "opened a window of opportunity" for obama to fix a problem that has long plagued both sides of the border.	34 %	16 %	32 %

Table 10.1 – Qualitative analysis of example 1 from data set.

Evaluating our results, none of summaries neither best ROUGE sentence make explicit what is the "problem that has long plagued both sides of the border", but we can infer that it is violence because of "connecticut school shooting" information. Extracting a simple sentence as summary ensures conciseness, but loses in completeness due to absence of key information. Example in Table 10.2 is about an incident with gunfire in Brazil, where a woman was shot and left unscathed with a bullet lodged in his bra. Ground-truth captures the key ideas in text. Evaluating our selected sentences, we observe that Recurrent Neural Network summarization result gives a richer context of situation, even presenting smaller ROUGE scores when compared to best sentence based on ROUGE scores. Vector Offset summarization method chooses the best sentence and consequently suffer from the same problem of completeness.

		ROUGE		
Method	Summary	1	2	L
Ground-truth	Ivete Medeiros was shot in the chest after being caught in robbery crossfire. After seeing no blood she thought bullet had missed her and thanked God. But then felt a ’burning sensation’ and discovered the bullet lodged in bra.	-	-	-
RNN	ivete medeiros, from the northern amazonian city of belém, was struck by the round as she left a supermarket to investigate commotion unfolding in the street outside.	20 %	3 %	19 %
Vector Offset	but she then felt ’a little burning sensation’ and after finding the source of the pain, discovered the bullet lodged in her underwear	47 %	19 %	39 %
Best Sentence	but she then felt ’a little burning sensation’ and after finding the source of the pain, discovered the bullet lodged in her underwear	47 %	19 %	39 %

Table 10.2 – Qualitative analysis of example 2 from data set.

11. RELATED WORK AND PERFORMANCE COMPARISON

The first automatic text summarization method date from 1958 [10]. Since then, many solutions have been developed applying the most diverse approaches, from statistical solutions to complex neural networks. In this Chapter we discuss solutions that influenced us and that use the same data set, allowing us to do performance comparison and discuss design choices.

11.1 Abstractive Summarization with Bidirectional RNN

Nallapati et al. [13] developed an abstractive text summarization model that uses an encoder-decoder architecture. This encoder is composed of a bidirectional GRU-RNN (Gated Recurrent Unit and Recurrent Neural Network units), while the decoder consists of a uni-directional GRU-RNN with the same dimensions of the encoder, an attention mechanism and a softmax layer to generate words based on a vocabulary.

Words are represented as word vectors with additional additional linguistic features such as parts-of-speech tags, name-entity tags, term-frequency and inverse document frequency. They include other features beyond word vectors representation, like term frequency and inverse document frequency, that is stored into an one-hot vector representation and used during training process.

Words out-of-vocabulary are usually replaced by an unknown tag. To solve this problem, this proposal implement a switch mechanism that decides between using a generated word in vocabulary or retrieving the original word from source document.

This model implement an attention model that operates independently at word level and sentence level, using two bidirectional recurrent neural networks, one for each level.

11.2 SummaRuNNER

SummaRuNNER is an extractive text summarization developed by Nallapati et al. [15]. This method uses two bidirectional Gated Recurrent Unit layers, one that run at word level and other at sentence level.

The decision if a given sentence must be included in the summary is done through a sigmoid activation function that classify each sentence. This decision is made based on content richness of sentence, its salience in relation to the document and its novelty in relation to accumulated summary representation.

This model needs a labeled data set to be trained. One problem related to training stage is that majority of documents only contain human written summaries, which requires the use of an unsupervised approach that convert abstractive summaries to extractive labels. This problem was solved later by employing an abstractive training method that uses a Recurrent Neural Network decoder that model the generation of abstractive summaries at training time.

Evaluation metrics are computed based on ROUGE scores with respect to ground-truth summaries. This method aim to maximize ROUGE score, but due to high computational cost, they adopt a greedy approach that add one sentence at a time to the summary and evaluate ROUGE score of current summary against ground-truth summary. The process of adding sentences to summary stops when none of remaining candidate sentences improve ROUGE score.

11.3 Performance Comparison

Performance comparison using ROUGE F-scores is shown in Figure 11.1, where we compare Vector Offset Summarization and Recurrent Neural Network RNN 1 with the models discussed earlier. Here, we verify that:

- Vector Offset Summarization surpasses the abstractive summarization model *words-lvt2k* (Section 11.1) in ROUGE-1 and ROUGE-2 F-scores, but is surpassed in ROUGE-L.
- SummaRuNNER surpasses all models, obtaining results close to best ROUGE scores in ROUGE-1 and ROUGE-2 and technically equals ROUGE-L.

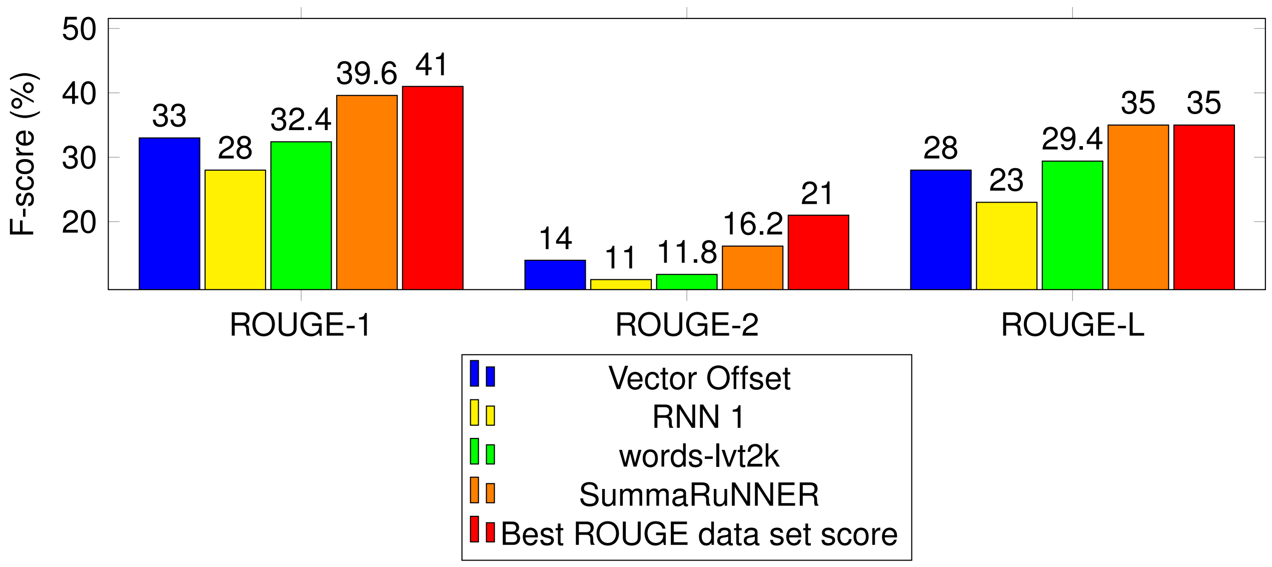


Figure 11.1 – Performance evaluation of our models with other researches.

Given this results, we believe that SummaRuNNER obtained state-of-the-art results because operates both at word level and sentence level, adding more layers of information to be processed and allowing a better fine-tuning of model parameters during training stage. SummaRuNNER also used a greedy algorithm to generate labels for supervised learning that is more complex than the one we used, what can drastically increase its ROUGE performance by generating best ground-truth summaries.

words-lvt2k is an abstractive summarization model that, due to rewriting the whole summary and avoiding reusing words of original text, presents lower ROUGE scores because of *n*-gram evaluation method.

12. CONCLUSION

During this work, we developed three text summarization methods and tested them over the news domain data set CNN and DailyMail: one method based on word vector offset, one based on Feed-Forward Neural Networks, and one based on Recurrent Neural Networks with long-term units.

Feed-Forward Neural Networks models, modeled as a classification problem, do not perform well in this task given our test scenario, in general suffering from overfitting.

Given the nature of texts, which are structured as a sequence of sentences and words, Recurrent Neural Networks seem a more suitable model. When we define our problem as a regression one using Recurrent Neural Network, the resulting model yields results close to Vector Offset Summarization. We use ROUGE F-scores as the output for sentences, but as demonstrated by other works, it is possible to include other elements to be taken into account in order to achieve performance improvement.

The Vector Offset Summarization method yields the best results of all our models, is simple to implement and requires low computational effort. Its results are close to state-of-the-art solutions and even surpass some abstractive summarization techniques for some evaluation metrics. Since this model is completely dependent upon the quality of the embeddings, the only way in which we envision improving this model, is to either use a higher-dimensional embedding (for a limited improvement), or refine the training of the embedding generator with more domain-specific data.

Automatic results evaluation solutions like ROUGE scores are time saving for analyzing large volumes of data, simple to implement and to understand how it works. This metric, however, is based on *n*-gram language model and expect exact matches when comparing sentences, ignoring the fact that natural languages allow distinct sentences to have the same semantic value, what can result in low scores for good summaries. During our qualitative analysis, we also observed that high ROUGE scores not necessarily mean a good summary in terms of completeness and conciseness. One alternative is developing an evaluation mechanism based on word vectors and distance between sentence vectors, given that word embeddings can capture semantic features of text.

Developing this work required the development of a set of skills that needed to be combined to accomplish our goals. Understanding Natural Language Models and especially neural network models, that include generating word vectors through a learning process and training neural networks to classify texts and to predict performance scores was challenging and exciting.

As next steps, improving Recurrent Neural Networks with long-term memory [5] seems to be the logical choice, specifically by implementing an encoder-decoder to accept variable length data as input, an attention model to retain information for longer time, and refining our data set to include other relevant information for each sentence, such as sentence length or presence of key words like nouns to bias the relevancy of sentences in relation to whole text. Developing automatic evaluation method based on word vectors is necessary to improve this task and overcome limitations imposed by *n*-gram models. Other desired application is developing a click bait detector by comparing the relevance of news headlines with its content.

REFERENCES

[1] João Paulo Aires, Juarez Monteiro, Roger Granada, and Felipe Meneguzzi. Norm conflict identification using vector space offsets. In *Proceedings of the 31st International Joint Conference on Neural Networks*, 2018.

[2] Sumit Chopra, Michael Auli, and Alexander M. Rush. Abstractive sentence summarization with attentive recurrent neural networks, 01 2016.

[3] Mahak Gambhir and Vishal Gupta. Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*, 47(1):1–66, Jan 2017.

[4] Jade Goldstein, Vibhu Mittal, Jaime Carbonell, and Mark Kantrowitz. Multi-document summarization by sentence extraction. In *Proceedings of the 2000 NAACL-ANLPWorkshop on Automatic Summarization - Volume 4*, NAACL-ANLP-AutoSum '00, pages 40–48, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[7] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015.

[8] Atif Khan. A review on abstractive summarization methods. 59:64–72, 01 2014.

[9] Chin-Yew Lin. Rouge: a package for automatic evaluation of summaries. July 2004.

[10] Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958.

[11] Tomas Mikolov, Scott Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics, May 2013.

[12] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[13] Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016.

[14] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features. In *NAACL 2018 - Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[15] Melrose Roderick, James MacGlashan, and Stefanie Tellex. Implementing the deep q-network. *CoRR*, abs/1711.07478, 2017.

[16] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.