

Réf : Ing-GI-2016-06

Rapport de Projet de Fin d'Études

Pour obtenir le

Diplôme d'Ingénieur en Génie Informatique

Option : GLID

Présenté et soutenu publiquement le 23 juin 2016

Par

Doghman MALEK

Sentiment Analysis using Neural Network

Composition du jury

Monsieur (Madame)	Souisi Emna	Président
Monsieur (Madame)	Kouki Zoulei	Rapporteur
Monsieur (Madame)	Dridi Seifeddine	Encadrant Entreprise
Monsieur (Madame)	Hacheni Narjes	Encadrant ENSIT

Année universitaire : 2015-2016

Contents

General Introduction	1
1 General Project Context	3
1.1 The Host company	3
1.1.1 Activity fields	3
1.2 Project's context	4
1.2.1 problematic	4
1.2.2 Project goals	4
2 Artificial Neural Networks	6
2.1 Introduction	6
2.2 Single Neuron	6
2.3 Artificial neural network	8
2.4 Architectures of neural networks	10
2.4.1 Recursive Neural Networks	10
2.4.2 Recurrent Neural Networks	10
2.4.3 Convolutional Neural Network	11
2.5 Training a neural network	13
2.5.1 Learning Paradigms	13
2.5.2 Supervised Learning	14
2.5.3 Unsupervised Learning	14
2.5.4 Reinforcement Learning	14
2.5.5 Learning Algorithm	14
2.5.6 Forward Pass	15

2.5.7	Output Layer	15
2.5.8	Cost function	16
2.5.9	Gradient Descent Algorithm	17
2.5.10	back-propagation	19
2.6	Conclusion	22
3	State of the art	23
3.1	Introduction	23
3.2	Sentiment Analysis	23
3.3	Vector representation of words	24
3.4	Word2Vec	25
3.5	Sentence Classification	29
3.5.1	Recursive Neural Tensor Network	29
3.5.2	Long Short Term Memory networks	30
3.5.3	Convolutional Neural Network	32
3.5.4	Neural Network Comparison Summary	35
3.6	Conclusion	35
4	Sentiment Analysis for Steam Games using CNN	37
4.1	Introduction	37
4.2	Approach Description	37
4.2.1	Preprocessing	38
4.2.2	Embedding layer	39
4.2.3	Convolution Layer	39
4.2.4	ReLU layer	40
4.2.5	Max pooling	40
4.2.6	The final output	40
4.2.7	Calculating the accuracy	41
4.2.8	Overfitting	41
4.3	Implementation	42
4.3.1	Production environment	42
4.3.2	Languages and Frameworks	42

4.3.3	Experiment and Result Analysis	45
4.3.4	Application	47
4.4	Conclusion	52
	Conclusion and Perspectives	53
	References	54

List of Figures

1.2	Logo de vNeuron	4
2.1	Single neuron	7
2.2	Common activation functions used in neural networks	8
2.3	A three-layer neural network with a single output	9
2.4	The tree structure of an RNN	10
2.5	An unrolled recurrent neural network revealing a chain-like nature. . .	11
2.6	The repeating module in a standard RNN contains a single layer. . .	11
2.7	Convolutional neural network Architecture.	12
2.8	A narrow convolution	12
2.9	The max pooling operation	13
2.10	3D Mean Squarre Error plot	17
2.11	The loss is decreasing after each step	18
2.12	Computation graph for the expression $e = (a + b) * (b + 1)$	20
2.13	the derivative of each node with respect to it's predecessor	21
3.1	Skip-Gram model	26
3.2	The probability that "Car" shows near "Mice"	27
3.3	T-SNE visualization of the learned word embeddings	28
3.4	Some learned semantic meaning between the word vectors	28
3.5	Constituency parse tree for the sentence " <i>A cat eats a mouse</i> "	29
3.6	A recursive neural network to predict sentence sentiment	30
3.7	Computing parent node using a compositionality function g	30
3.8	An LSTM cell	31
3.9	LSTM used for binary sentence classification	32

3.10	Convolutional Network architecture for sentence sentiment	33
3.11	A max pooling operation resulting in dimension reduction while con- serving important information	34
3.12	Neural Network Comparison Summary	35
4.1	The CNN architecture used in our project	38
4.2	Accuracy for the training set (blue plot) and the test set (red plot) .	46
4.3	The model overfitting the data	46
4.4	The loss curve after applying regularization methods	47
4.5	The accuracy curve after applying regularization methods	47
4.6	Home page	48
4.7	The model accurately predicting the positive sentiment	49
4.8	the model predicted a very positive sentiment for the game Dota 2 with an accuracy of 82%.	50
4.9	A noisy positive comment	51
4.10	Another noisy positive comment	51
4.11	The model accurately predicting a mixed sentiment for the AAA game Assassin's Creed Unity with an accuracy of 94%.	52

List of Algorithms

1	Gradient Descent Algorithm	19
2	Gradient Descent and Backpropagation in a Neural Network	22
3	Skip-Gram Word2Vec	27

General Introduction

Introduction

Recent years have brought the burst of popularity of community portals across the Internet. Internauts provide their input not only through discussions and personal notes in various social web spaces (boards, blogs, facebook, twitter etc) but also on mass scale leave comments and reviews of products and services on numerous commercial websites (Amazon, Yelp, Steam etc). The fast growth of such content has not been fully harnessed yet.

Sentiment Analysis (also referred as Opinion mining), a subfield of natural language processing (NLP), is an attempt to take advantage of the vast amounts of user generated content. It employs some machine learning techniques and natural language processing to formalize the knowledge taken from user opinions and analyze it for further reuse. Over the last decade a huge increase of interest in the sentiment analysis research is clearly visible.

Nowadays, companies are in need for intelligent systems that can analyse reviews collected from their customers in order to detect whether they are satisfied or not. It has been known that the task of classifying clients opinions into negative and positive can be challenging. Some cases involve the understanding of negation words, their scope, and other semantic effects.

In the following work, we will analyse the sentiment of the Steam games reviews using Artificial Neural Networks (ANNs). The ANN is a supervised machine learning technique used for classification. It is a model inspired by biological neural networks in the human brain. The ability to split the large amount of data between its neuron makes the training

Document Structure

The chapters are grouped into four parts: the host company is presented in the first chapter, background material is presented in the second chapter, the next chapter present the state of the art in sentiment analysis and the last chapter describes the method used in our project.

The second chapter introduces a very powerful model in machine learning that is the artificial neural networks, their structure and how they are trained.

The third chapter present an overview of the state of the art in the area of sentiment analysis and describes some deep learning model used for opinion mining based on the architectures described in the first chapter. It also provides a comparison study.

In the fourth chapter, we will explain the deep learning model used in the current work, we will detail the hard and software environment and we will provide some screen-shots showing the web application interfaces.

Chapter 1

General Project Context

1.1 The Host company

IP-Tech was founded in 2007, now it is a group of enterprises. It comprises:

- IP-Tech.
- AchieveYourDocs.
- vNeuron.

1.1.1 Activity fields

IP-Tech is a nearshore and offshore Tunisian IT services company and also a software editor in the field of Document Management System (DMS). AchieveYourDocs is a subsidiary company of IP-Tech founded in 2009 specialized in the Electronic document management and the editor of Averroes a Document Management System (DMS), Entreprise Content Management (ECM) and Business Process Management (BPM). It offers the possibility to:

- Archive, share and search documents.
- provide APIs and web services to facilitate the integration with any existing system.
- and automate document work-flows.



Another subsidiary company of IP-Tech is vNeuron founded in 2015. It is specialized in big data analytics, Artificial intelligence techniques and Machine learning.



Figure 1.2: Logo de vNeuron

1.2 Project's context

1.2.1 problematic

Sentiment analysis (also known as opinion mining) refers to the use of natural language processing and machine learning to identify and extract subjective information in source materials. Sentiment analysis is widely applied to reviews and social media for a variety of applications, ranging from marketing to customer service.

Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic and predicts the polarity of given comments. For this classification task, several supervised machine learning solutions are provided, ranging support vector machine Classifiers to artificial neural networks.

1.2.2 Project goals

The goal of this project is to design a neural network model to analyse the sentiment i.e to predict the polarity of given comments and classifying them into positive and

negative. The solution must respond quickly and achieve high accuracy.

Chapter 2

Artificial Neural Networks

2.1 Introduction

Artificial neural networks (ANNs) are a family of models inspired from the biological neural networks, in particular the human brain. They are well suited to Machine learning problems especially when the input is very large.

The first section will explain the basic concept of a single neuron and then we will generalize to a network of neuron. The next section will present some common architecture and in the final section we will show how training the network is done.

2.2 Single Neuron

To describe neural networks, we will begin by the simplest possible neural network, one which comprises a single "neuron."

A neuron is a computational unit that takes an input $x \in \mathbb{R}^n$ and outputs a value \hat{y} :

$$\hat{y} = f(wx + b) \in \mathbb{R}^n \quad (2.2.1)$$

f is called the activation function, $w \in \mathbb{R}^n$ is the weight and $b \in \mathbb{R}$ is the bias [1].

Each w_i is the weight of the input x_i , it denotes how much that specific input impacts the output. The figure 2.1 represents a single neuron diagram.

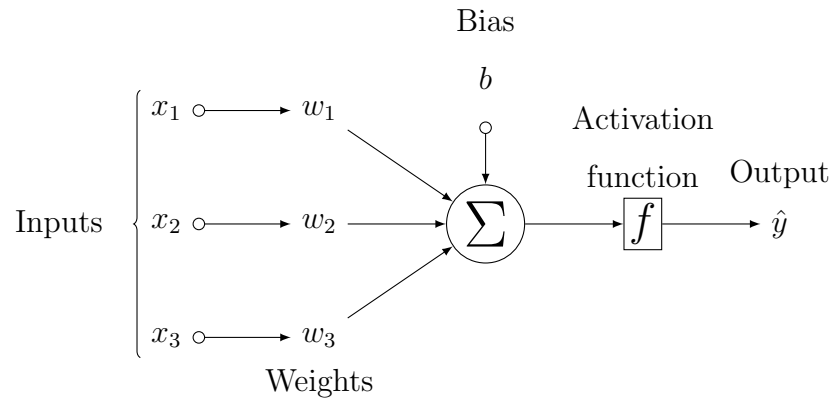


Figure 2.1: Single neuron

As we can see, the neuron perform three operations.

- **Weighting:** each x_i is multiplied by its corresponding weight w_i .
- **Sum:** The neuron sums the weighted inputs and adds the bias b .
- **Activation:** The value of \hat{y} is calculated.

Some common activator functions are the linear function, tanh, ReLU and the sigmoid function. Because the former is very limited (it can only learn linearly separable points), non linear functions are used instead.

Some desirable properties (some are not necessary) in an activation function are:

- **Nonlinear:** When the activation function is non-linear, then a two-layer neural network can be proven to be a universal function approximator [2].
- **Continuously differentiable:** This property is necessary because it enables the neuron to learn using the gradient-based optimization methods such as the gradient descent algorithms that we will encounter in next sections.
- **Range:** When the range of the activation function is finite, gradient-based training methods tends to be more stable.
- **Monotonic:** When the activation function is monotonic, the error surface is guaranteed to be convex [3].

The figure 2.2 shows some activation functions, their mathematical equation and their graphical representation.

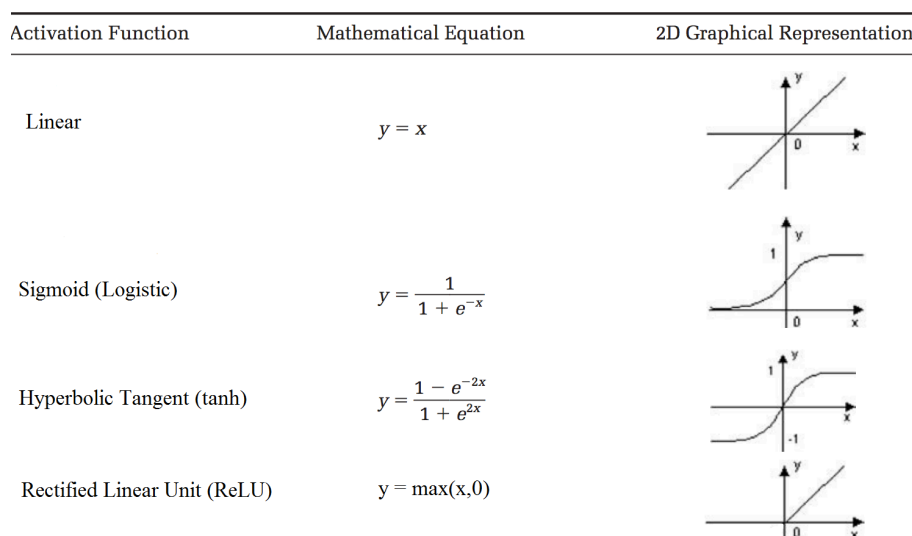


Figure 2.2: Common activation functions used in neural networks

2.3 Artificial neural network

A neural network is formed by joining together many of our simple "neurons," so that the output of a neuron can be the input of another. The diagram 2.3 represent a simple neural network with three layers. The depth of a neural network is the number of its layers [1].

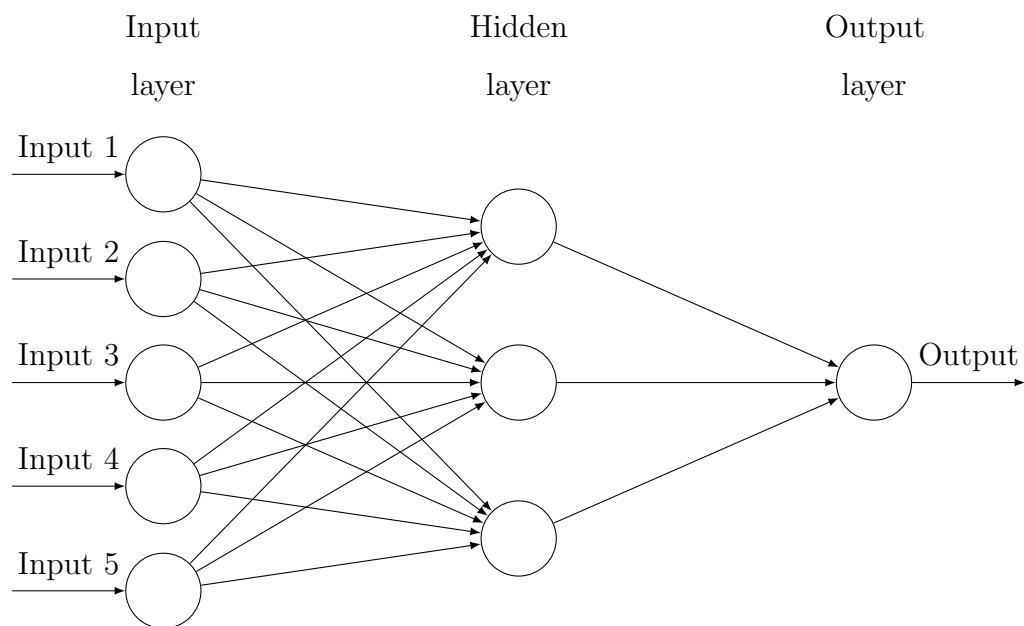


Figure 2.3: A three-layer neural network with a single output

- **input layer:** The leftmost layer of the network, the nodes receive the input and pass it to the hidden layers
- **hidden layer(s):** It is just one layer in this example but they can be much more. The neurons in each hidden layer have the same activation function. The calculated output is sent to the next layer.
- **output layer:** the rightmost layer, It have one node but they can be more.

In this example each neuron is connected to all the neurons in the next layer, we say that our network is fully connected. Such architecture is called a feedforward network or multi-layer perceptron MLP. This need not be the case, other networks don't follow this architecture. When the depth of a neural network exceeds three layers, it becomes a deep neural network.

2.4 Architectures of neural networks

2.4.1 Recursive Neural Networks

Recursive Neural Networks (RNNs) are a kind of deep neural network created by applying the same weights w over a tree structure.

The representation of each parent node p is a combination of it's children x_a and x_b using a shared weight matrix and a non-linear transformation such as \tanh e.g

$$p = \tanh(W[x_a; x_b]) \quad (2.4.2)$$

where w is a $[n, 2n]$ matrix, x_a and x_b are k -dimensional vector.

p can also be written as

$$p = \tanh(W_L x_a + W_R x_b) \quad (2.4.3)$$

where W_L and W_R are $[n, n]$ learned matrix. The figure 2.4 shows the most simple architecture for an RNN.

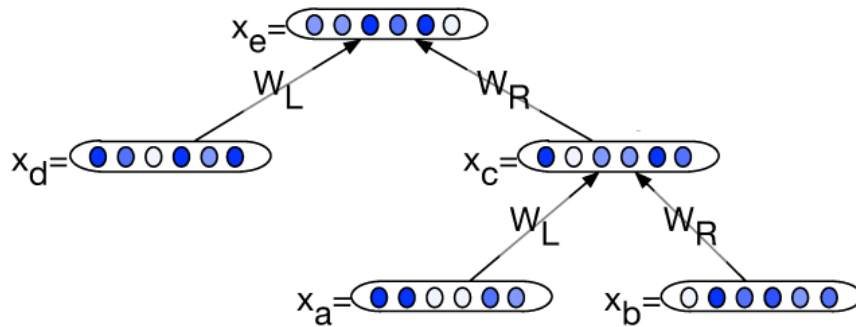


Figure 2.4: The tree structure of an RNN

RNNs have been successfully applied to model compositionality in natural language using parse-tree-based structural representations.

2.4.2 Recurrent Neural Networks

Recurrent Neural Networks (also called RNNs, not to be confused by recursive neural networks) are in fact a particular recursive neural network with a linear chain-like structure [4].

They are networks with loops in them, each neuron send its outputs to the next neuron, this way each neuron have some information about previous neurons. This is how a recurrent neural network look like.

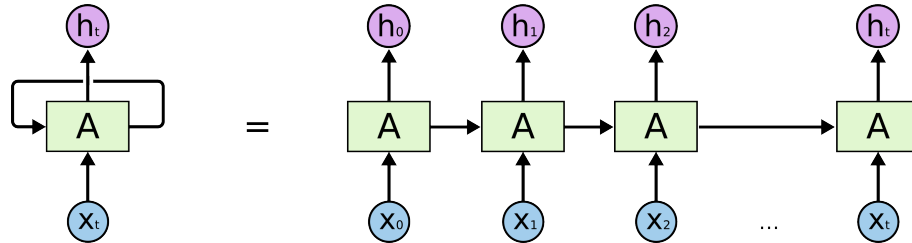


Figure 2.5: An unrolled recurrent neural network revealing a chain-like nature.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data. The repeated module can have a very simple structure such as a simple *tanh* layer.

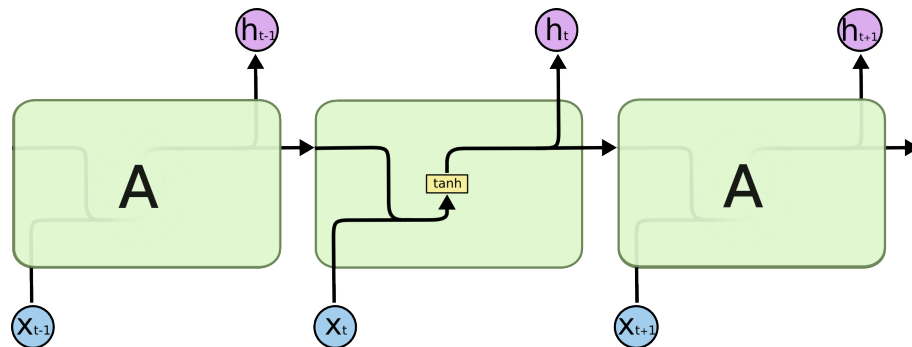


Figure 2.6: The repeating module in a standard RNN contains a single layer.

Recurrent networks are well suited for NLP tasks, for speech recognition and for translation.

2.4.3 Convolutional Neural Network

Convolutional Networks (CNNs, also called ConvNets) are mainly invented for computer vision where the input consists of images. That's why, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth as illustrated by the figure 2.7. Note that the word depth here refers

to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.

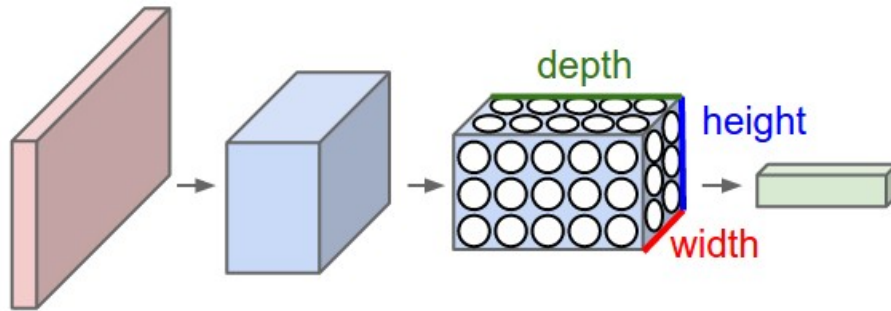


Figure 2.7: Convolutional neural network Architecture.

It arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels) [5].

In each step, successive layers of convolutions and pooling are used, this way the width and the height are reduced while the depth is increased to finally obtain an n-dimension vector called the feature vector.

- Convolution:

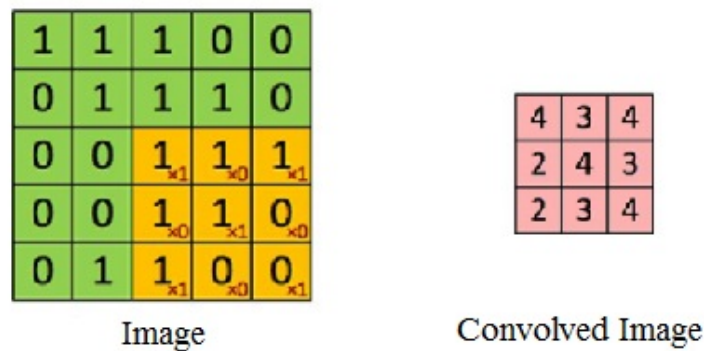


Figure 2.8: A narrow convolution

The convolution is a sliding window also called filter over the image. In the figure 2.8 the filter is a 3x3 matrix (yellow square), the values in this matrix are called the weights. The convolution is used to reduce the dimension of the input. Intuitively the convolution only extract the most relevent feature from the input and discard non important information.

- Pooling:

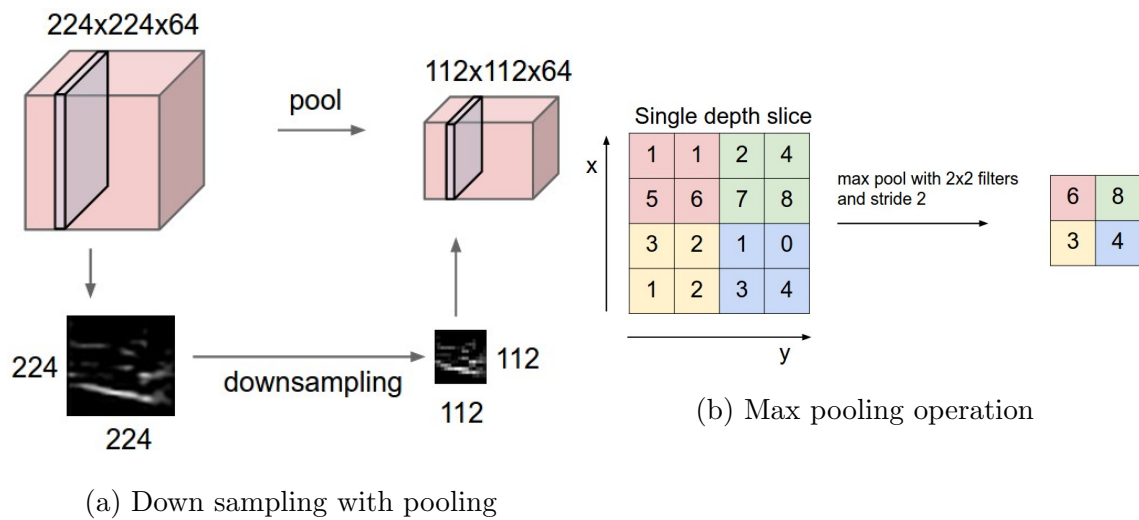


Figure 2.9: The max pooling operation

Like convolution the pooling is used to reduce dimensionality while preserving the depth. It is applied the same way as the convolution but it doesn't have any weights. In the end, CNNs have wide applications in image and video recognition [6], recommender systems [7] and recently in natural language processing [8].

2.5 Training a neural network

This section describes the learning paradigms used in machine learning then it will explain how to train neural networks using the gradient descent algorithm.

2.5.1 Learning Paradigms

There are three major learning paradigms, each corresponding to a particular abstract learning task. These are supervised learning, unsupervised learning and rein-

forcement learning.

2.5.2 Supervised Learning

Machine learning problems where a set of input-target pairs is provided for training are referred to as supervised learning tasks. A supervised learning task consists of a training set S of input-label pairs (x, y) . The goal is to use the training set to output a predicted value \hat{y} and then minimise some task specific error measure J between the label y and this predicted value \hat{y} [9].

2.5.3 Unsupervised Learning

Unsupervised learning, where no training signal y exists at all, the algorithm attempts to uncover the structure of the data set S consisting of inputs x by inspection alone. Since no labels are given, there is no cost function to minimize.

This paradigm is mainly used for data clustering i.e grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other clusters.

An unsupervised task used in the current work is the vector representation of the words explained in the section 3.4.

2.5.4 Reinforcement Learning

Reinforcement Learning allows machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behaviour; this is known as the reinforcement signal.

2.5.5 Learning Algorithm

We will not consider either reinforcement learning or unsupervised learning, we will only cover the supervised training of a feedforward network. Training other network architecture is roughly the same. Training a neural network is done in four major steps:

- Forward Pass.
- Output the predicted vector.
- Computing the loss compared with the label vector.
- Backpropagation and gradient descent algorithm.

2.5.6 Forward Pass

Consider an MLP with I input units, activated by input vector x . Each unit in the first hidden layer calculates a weighted sum of the input units. For hidden unit h , we refer to this sum as the network input to unit h , and denote it a_h . The activation function f_h is then applied, yielding the final activation b_h of the unit. Denoting the weight from unit i to unit j as w_{ij} , we have

$$a_h = \sum_{i=1}^I w_{ih} x_i \quad (2.5.4)$$

$$b_h = f_h(a_h) \quad (2.5.5)$$

A forward pass is calculating all the activation of the neural network unit. Each neuron sends its output activation b_h to the neuron in the next hidden layer. This way, for a unit h in the hidden layer l the output is computed as follow:

$$a_h = \sum_{h'=1}^{H_{l-1}} w_{h'h} b_{h'} \quad (2.5.6)$$

$$b_h = f_h(a_h) \quad (2.5.7)$$

2.5.7 Output Layer

The output vector \hat{y} of an MLP is given by the activation of the units in the output layer. The network input a_k to each output unit k is calculated by summing over the units connected to it, exactly as for a hidden unit, i.e.

$$a_k = \sum_{h=1}^{H_m} w_{hk} b_h \quad (2.5.8)$$

for a network with m hidden layers.

Both the number of units in the output layer and the choice of output activation function depend on the task. For binary classification tasks, the standard configuration is a single unit with a logistic sigmoid activation. Since the range of the logistic sigmoid is the open interval $(0, 1)$, the activation of the output unit can be interpreted as an estimate of the probability that the input vector belongs to the first class (and conversely, one minus the activation estimates the probability that it belongs to the second class).

For classification problems with $K > 2$ classes, the convention is to have K output units, and normalise the output activations with the softmax function to obtain estimates of the class probabilities. The Softmax function squashes a K -dimensional vector of arbitrary real values to a K -dimensional vector $\sigma(z)$ of real values in the range $(0, 1)$ that add up to 1. It is given by the formula 2.5.9:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad (2.5.9)$$

A 1-of- K coding scheme also called One-hot vector represent the target class z as a binary vector with all elements equal to zero except for element k , corresponding to the correct class C_k , which equals one. For example, if $K = 5$ and the correct class is C_2 , z is represented by $(0, 1, 0, 0, 0)$.

2.5.8 Cost function

After computing the output represented by a k -dimensional vector \hat{y} , we need to measure the error between the predicted output \hat{y} and the correct label y given by the One-hot vector. A very common error function (also called cost function) is the mean square error.

$$MSE = J(w, b) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (2.5.10)$$

There is a sum in the cost function because in general the loss is calculated for a batch of data not a single example. Choosing the adequate loss function for each ML problem is important. For now we will use the MSE defined in 2.5.10 as a cost function.

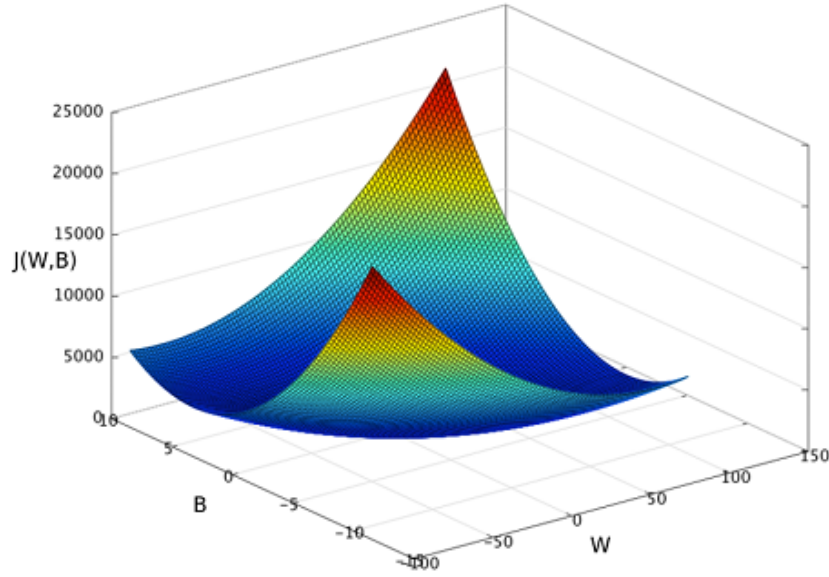


Figure 2.10: 3D Mean Squarre Error plot

The goal is to find the optimal values of w and b in order to minimize the loss function J , for this task we are going to use the gradient descent algorithm.

2.5.9 Gradient Descent Algorithm

Finding the minimum of the cost function $J(w, b)$ is done by calculating it's gradient over w and over b . Then one takes steps proportional to the negative gradient. Updating the parameters will be as follow.

$$w \leftarrow w - \gamma \frac{\partial J}{\partial w} \quad (2.5.11)$$

$$b \leftarrow b - \gamma \frac{\partial J}{\partial b} \quad (2.5.12)$$

Where γ a small constant called the learning rate

$$\text{and } \frac{\partial J}{\partial w} = \frac{2}{n} \sum_{i=1}^n -x_i(y - \hat{y}_i) \quad (2.5.13)$$

$$\text{and } \frac{\partial J}{\partial b} = \frac{2}{n} \sum_{i=1}^n -(y - \hat{y}_i) \quad (2.5.14)$$

Our training then consists of several steps where each time we take some input x , compute the loss and update our parameters until the system converge to the best values.

This process is known as gradient descent. We can take a look at this figure to visualize how the loss is decreasing after each step for the previous example.

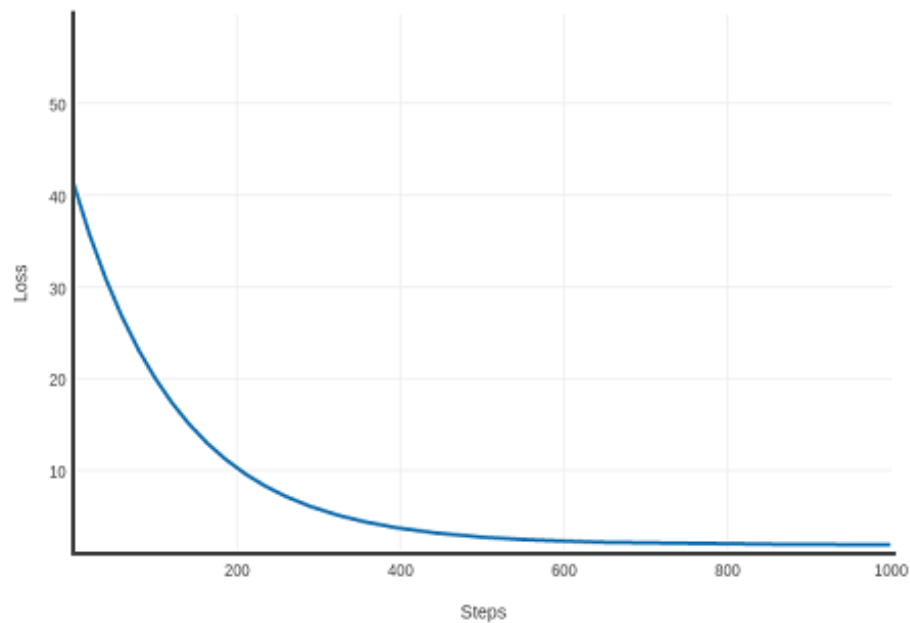


Figure 2.11: The loss is decreasing after each step

This is the gradient descent algorithm for minimizing the loss and updating the value of w . We do the same with the b parameter.

Algorithm 1 Gradient Descent Algorithm

```

1: Begin
2: Initialization:
3:  $w_{new} \leftarrow \text{random}()$ 
4:  $w_{old} \leftarrow 0$ 
5:  $\gamma \leftarrow 0.01$ 
6:  $\text{precision} \leftarrow 0.0001$ 
7: Steps:
8: do
9:    $w_{old} \leftarrow w_{new}$ 
10:   $w_{new} \leftarrow w_{old} - \gamma \frac{\partial f(w_{old})}{\partial w}$ 
11: while  $|w_{new} - w_{old}| \geq \text{precision}$ 
12: return  $w_{new}$ 
13: End

```

Finally, these are some important additional concept about the gradient descent.

- **Convexity:** The MSE cost function used is convex. Regardless of how we initialized our parameters, we will end up at the same absolute minimum. That's why, in most cases, a convex error function is used to compute the loss.
- **Performance:** Lowering too much the learning rate γ , will result in better accuracy but a slower convergence to the minimum. Greater steps can cause the system to oscillate around the minimum. As a solution, we can exponentially decrease γ , starting with long steps and ending with smaller steps as we approach the minimum.
- **Stopping Criteria:** we haven't defined when to stop searching for a solution. In the previous example we just did 1000 iterations. But this is typically done by looking at the small variation in the loss.

2.5.10 back-propagation

A key mathematical insight in NNs is the chain rule, it is very effective and it makes the training of the network very fast. If two functions gets composed then the chain

rule tells us that the derivative of that function is the product of the derivative of each component. It has this formula:

$$[f(g(x))]' = f'(g(x)) \cdot g'(x) \quad (2.5.15)$$

The chain rule may be written, in Leibniz's notation, in the following way. We consider z to be a function of the variable y , which is itself a function of x (y and z are therefore dependent variables), and so, z becomes a function of x as well:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \quad (2.5.16)$$

To see how this formula would work in practice, let's look at the computational graph of the expression $e = (a + b) * (b + 1)$.

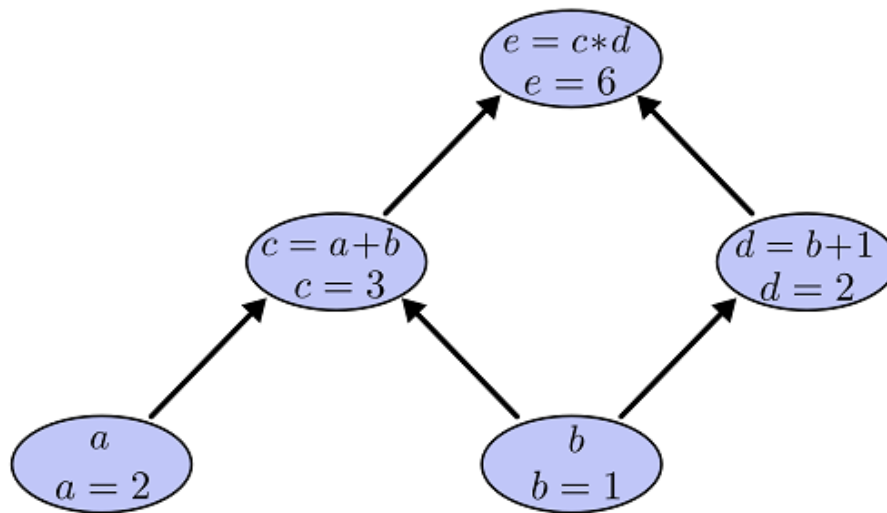


Figure 2.12: Computation graph for the expression $e = (a + b) * (b + 1)$

We can evaluate the expression by setting the input variables to certain values and computing nodes up through the graph. For example, setting $a = 1$ and $b = 2$, evaluate the expression to 6. Now let's calculate the gradient of e with respect to each node.

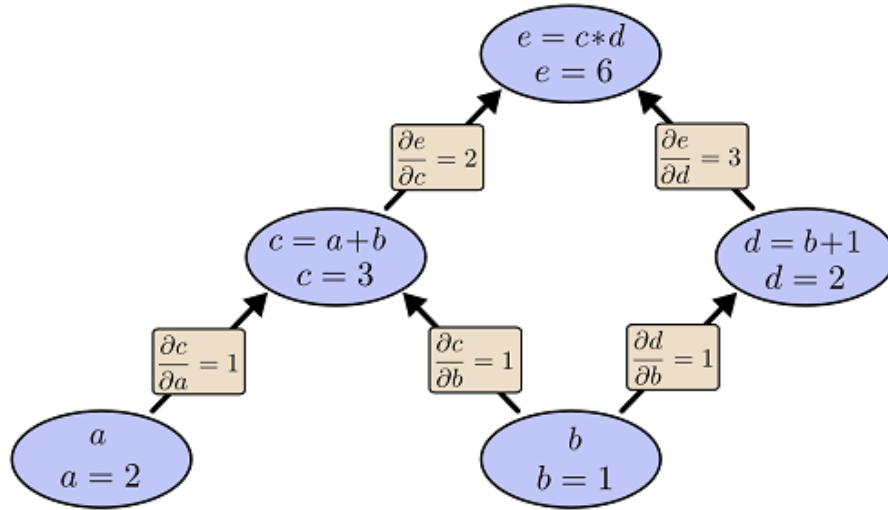


Figure 2.13: the derivative of each node with respect to it's predecessor

To get the derivative of one node with respect to another, The general rule is to sum over all possible paths from that node to the other, multiplying the derivatives on each edge of the path together. For example the derivative of e with respect to b is :

$$\frac{\partial e}{\partial b} = 1 * 2 + 1 * 3 \quad (2.5.17)$$

Our network is exactly the same as a computational graph. First there is a forward propagation to compute all activations through the network including the final output \hat{y} then there is a back-propagation to compute the gradient and update the parameters of each node [10].

Below is the gradient descent algorithm in neural networks. This algorithm is the same as the previous one but this time the gradients are computed using the back-propagation. w_k^L and b_k^L denotes the parameters of the node k in the layer L .

To resume, the neural network is feeded with data in the forward pass and it adjust its parameters to learn it.

Algorithm 2 Gradient Descent and Backpropagation in a Neural Network

```

1: Begin
2: Initialization:
3:  $w_k^L \leftarrow \text{random}()$ 
4:  $b_k^L \leftarrow 0$ 
5:  $\gamma \leftarrow 0.01$ 
6: do
7: Forward Pass:
8:   Compute the output of each node.
9: Backpropagation to compute the gradients:
10:  for each node  $k$  in layer  $L$  in the network do
11:    Compute  $\nabla w_k^L \leftarrow \frac{\partial f_k^L(w_i)}{\partial w}$ 
12:    Compute  $\nabla b_k^L \leftarrow \frac{\partial f_k^L(b_i)}{\partial b}$ 
13:  Update the parameters:
14:     $w_k^L \leftarrow w_k - \gamma \nabla w_k^L$ 
15:     $b_k^L \leftarrow b_k - \gamma \nabla b_k^L$ 
16: while Stopping criteria not met
17: End

```

2.6 Conclusion

In this section we explained the basics concepts behind neural network and how they are trained using the gradient descent algorithm combined with Back-propagation.

In the next chapter, we are going to inspect some deep learning models used in sentiment analysis.

Chapter 3

State of the art

3.1 Introduction

This chapter is composed of three parts. The first is a brief introduction to sentiment analysis, the second part is about an unsupervised learning task called vector representation of words. The final part will describe the models used in our task.

3.2 Sentiment Analysis

A basic and naive approach in sentiment analysis would be to look in isolation at each word in the sentence, attributing a positive score if it is positive and a negative score otherwise. That way, the order of the words is ignored, important information is lost and the machine will be easily fooled.

For example, because "witty" and "funny" are positive words, a naive approach would rate the following sentence as positive.

This movie was actually neither that funny, nor super witty.

Deep learning models have achieved remarkable results in computer vision [6] and speech recognition [11] in recent years. Within natural language processing, much of the work with deep learning methods has involved learning word vector representations through neural language models [12, 13, 14] and performing composition over the learned word vectors for classification [15]. As those representations gave a significant improvement in many NLP tasks, they became a main component.

3.3 Vector representation of words

For tasks like object or speech recognition all the information required to successfully perform the task is encoded in the data e.g an image is a matrix of individual raw pixel-intensities [16].

However, because we can't feed a word as a text string to a neural network, natural language processing systems traditionally treat words as discrete atomic symbols, and therefore 'cat' may be represented as Id537 and 'dog' as Id143. These encoding are arbitrary, and provide no useful information to the system regarding the relationships that may exist between the individual symbols [16].

This means that the model can leverage very little of what it has learned about 'cats' when it is processing data about 'dogs' (such that they are both animals, four-legged, pets, etc.). Representing words as unique, discrete Ids furthermore lead to data sparsity, and usually means that we may need more data in order to successfully train our models.

So to overcome these limits, we're going to turn to unsupervised learning to determine these vector representations from large raw text. Vector space models (VSMs) represent words in a continuous vector space where semantically similar words are mapped to nearby points. VSMs have a long, rich history in NLP, but all methods depend in some way or another on the Distributional Hypothesis, which states that words that appear in the same contexts share semantic meaning. For example, let's take a look at the following sentences.

The cat purrs

The cat hunts mice

In these sentences, it is perfectly reasonable to say the "kitty" purrs or this "kitty" hunts mice. The context words (colored in green) gives us a strong idea that "cat" and "kitty" are very similar. So we don't really have to know what the words actually means, we just get the meaning from its associated context.

Then "cat" is mapped to a vector V_c and "kitty" to a vector V_k , the similarity is calculated using the cosine function between the two vectors. In this case it should be close to 1.

$$\cos \theta = \frac{V_c \cdot V_k}{\|V_c\| \cdot \|V_k\|} \quad (3.3.1)$$

Now that we understood the basic idea behind the words embedding, lets get an intuitive feel for how this would work in practice. We will mainly look at the model proposed by *Mikolov et al* [14] named word2vec.

3.4 Word2Vec

Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text. It comes in two flavors, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model [14]. Algorithmically, these models are similar, except that CBOW predicts target words (e.g. 'mice') from source context words ('the cat hunts'), while the skip-gram does the inverse and predicts source context-words from the target words. The skip-gram model tends to perform better in larger datasets [16].

Because we dont have labels in this unsupervised learning task, we will create them ourselves from the raw text. For example, lets consider this dataset which consists of just a sentence.

The little cat hunts mice.

From this data, we first form a set of pairs of words and the context in which they appear. The context is a fixed size window, e.g our window is the n words to the left to a given word and n ones to the right.

Using a the window size $n = 1$ we have this dataset of (target,context) pairs.

$[Little, (the, cat)], [cat, (little, hunts)], [hunts, (cat, mice)]$

The skip-gram model takes the target word as input and tries to predict the context, for example given the target word "cat" the system will try to predict "little" and "hunts".

The Skip-Gram model is a feed-forward neural network. The input layer is represented using the one-hot vector representation described in the section 2.5.7.

In our case this vector will have n component where n is the vocabulary size. The hidden layer is represented by a weight matrix $w \in \mathbb{R}^{nm}$, m is the size of the word vector that we want to learn. When input vector is multiplied by the weight matrix w , we will select the matrix row corresponding to the 1 our target word vector embedding V [17]. The figure 3.1 represents the architecture of the skip-gram model.

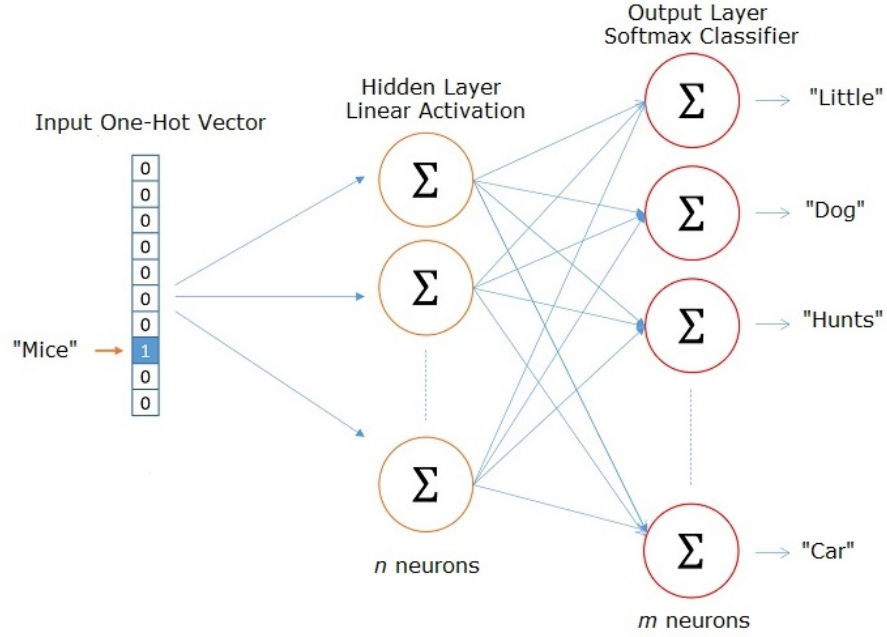


Figure 3.1: Skip-Gram model

Finally, The output layer of the network is a single n -dimensional vector containing, for every word in our vocabulary, the probability that each word would appear near the input word. The output value in each output neuron i is computed as described in the equation 3.4.2 where w_i^T denotes the transpose of the row i in w .

$$\hat{y}_i = \sigma(Vw_i^T) \quad (3.4.2)$$

The computing of the output value is illustrated in the figure 3.2

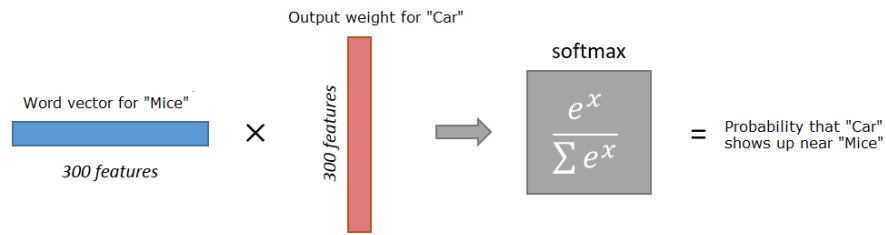


Figure 3.2: The probability that "Car" shows near "Mice"

Also the pseudo-code for this model is described in the algorithm 3.

Algorithm 3 Skip-Gram Word2Vec

```

1: Begin
2: do
3:   Embedding:
4:     Embed the target word into a vector  $V$ 
5:   Linear Model:
6:     Predict context words from the target word using  $f(v) = wv + b$ 
7:   Calculate the loss:
8:     Compute the loss between predicted context words and the correct one
9:   Update the embedding vector:
10:    Tweak  $V$  to minimize the loss
11: while Stop Criteria not met
12: End
  
```

After the training, we can visualize our learned vectors using a method called t-SNE, an algorithm to reduce vectors dimension non linearly to 2 dimensions [18]. Examining the plot, we can see that neighboring words tend to be related [19].

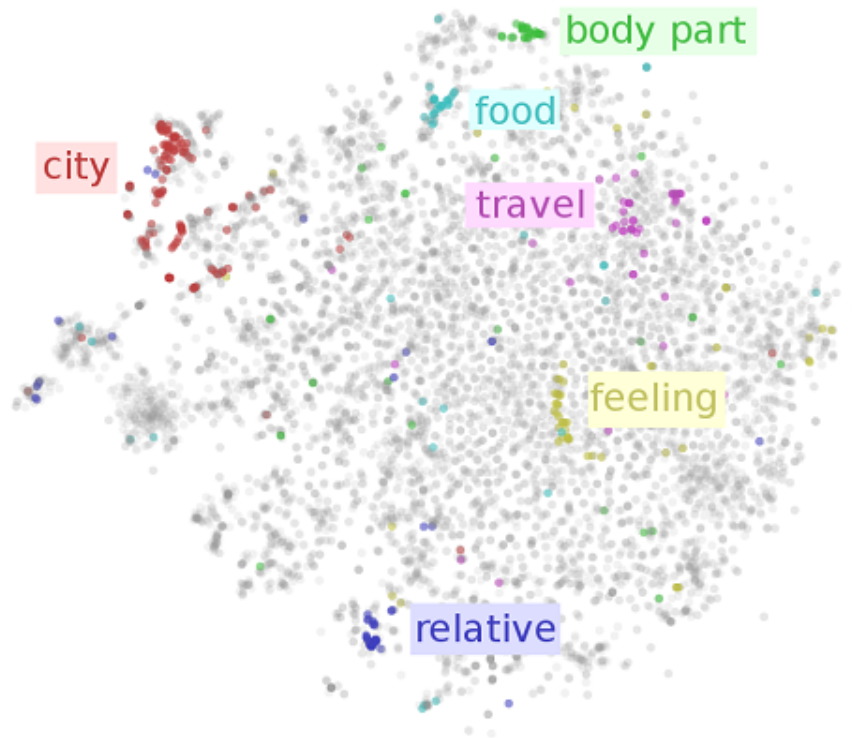


Figure 3.3: T-SNE visualization of the learned word embeddings

Even further, it appears that the vectors capture some general and in fact quite useful, semantic informations about the words and their relationships with one another. It seems that the information is encoded in the difference between the word vectors [20].

$$V_{king} - V_{queen} \simeq V_{man} - V_{woman}$$

$$V_{Spain} - V_{Madrid} \simeq V_{Japan} - V_{Tokyo}$$

This is illustrated in the figure 3.4 below.

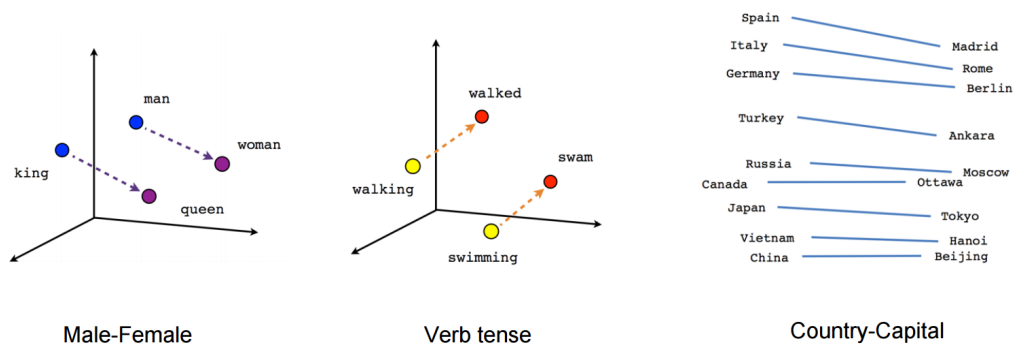


Figure 3.4: Some learned semantic meaning between the word vectors

3.5 Sentence Classification

In this section we are going to present some deep learning models for analyzing sentences.

3.5.1 Recursive Neural Tensor Network

Recursive Neural Tensor Network (RNTN) was developed for natural language processing. They have a tree structure with a neural net at each node [21].

The first step to build this neural network is to use word vectorization, which can be accomplished with the word2vec algorithm explained in previous sections. It creates a look-up table that will supply the vectors when needed.

To organize the sentences, RNTN uses constituency parse tree. This method breaks a text into sub-phrases. The non-terminal nodes represent the types of the phrases while the terminals represent the words in the sentence e.g noun phrases NP and verb phrases VP. Later these trees are binarized making each parent node exactly have two child leaves. This way, the sentences are a sort of trees having their root at the top and leaves at the bottom. Each sentence will look like the figure 3.5.

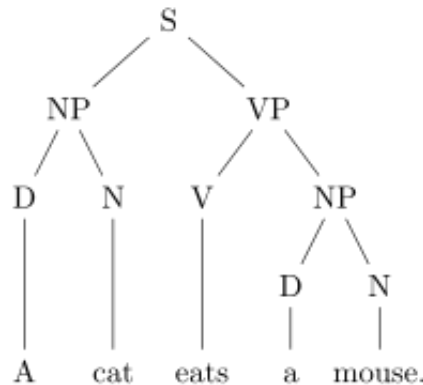


Figure 3.5: Constituency parse tree for the sentence "A cat eats a mouse"

For the classification task, *Socher et al* [21] introduced the Stanford sentiment tree bank with fully labeled parse tree for a complete analysis of the compositional semantic effects.

In the figure 3.6, each node in this tree denotes the polarity of its word, i.e "0" means that the word is neutral while "+" means that the word is positive.

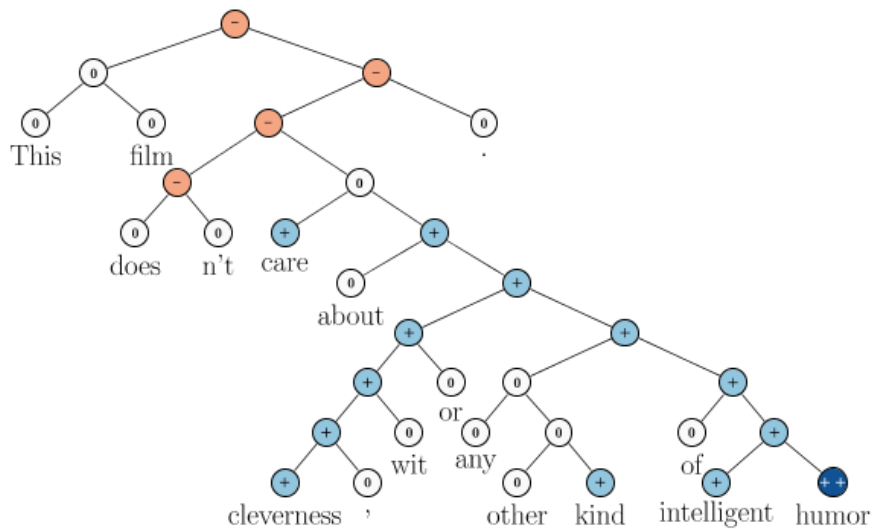
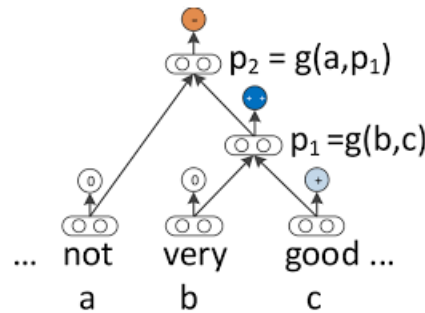


Figure 3.6: A recursive neural network to predict sentence sentiment

When an n-gram is given to the model, it is parsed into a binary tree where each leaf corresponding to a word is represented as a vector, the model will first compute the parent node vector in a bottom up fashion from the leaves then it will climb to the tree root using different types of compositionality functions g . For the ease of exposition, this is how the computing is done for a tri-gram.

Figure 3.7: Computing parent node using a compositionality function g

3.5.2 Long Short Term Memory networks

Long Short Term Memory networks (LSTMs) are networks with loops in them, each neuron send its outputs to the next neuron, this way each neuron have some information about previous neurons. this chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They are the natural

architecture of neural network to use for such data [4].

A special kind of RNN are the Long Short Term Memory or just LSTMs. While a classic RNN cell just contain a single tanh layer, the LSTM cell is composed of four non-linear layers, a tanh layer and three sigmoid layers. It also has two outputs : h_t and C_t .

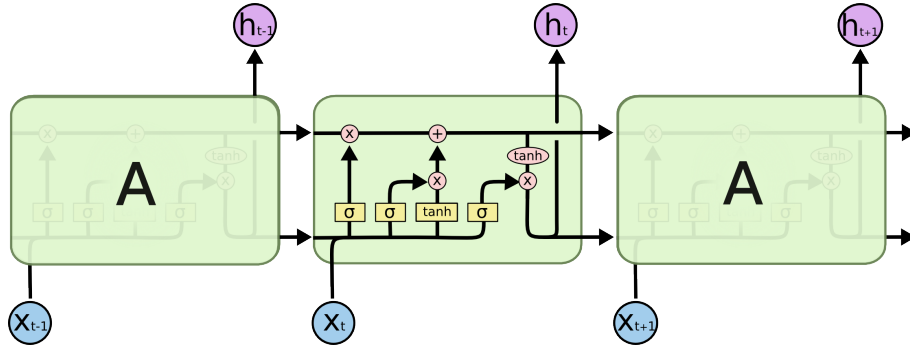


Figure 3.8: An LSTM cell

To keep it simple, LSTM were desinged to prevent the gradient vanishing. Doing back-propagation in classic RNN chains, ends up with the gradient being too small or zero because we are applying the gradient many times over the same entity.

For the task of sentence classification, we just add a mean pooling layer in top of the LSTM chain ending with a softmax classifier. The architecture of this model [22] is shown in the figure 3.9.

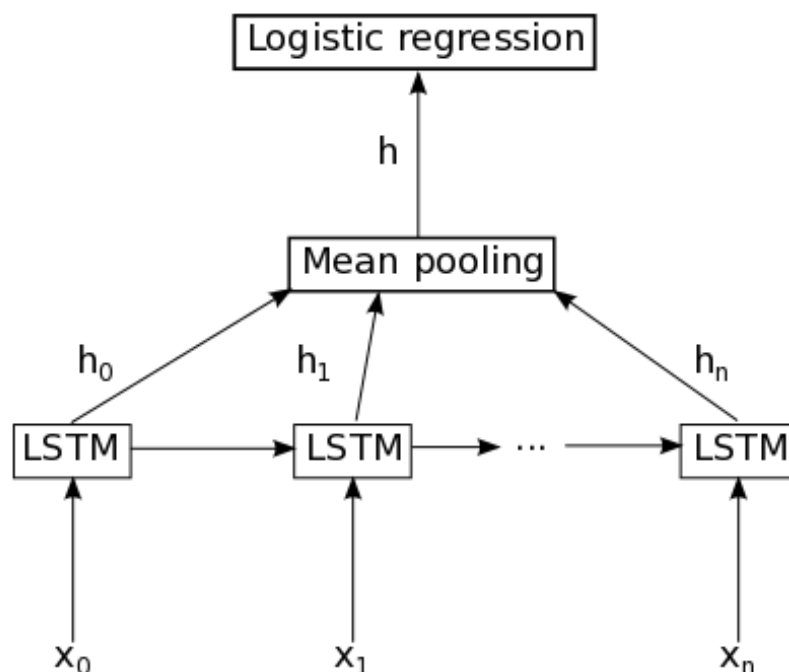


Figure 3.9: LSTM used for binary sentence classification

3.5.3 Convolutional Neural Network

The Convolutional Network (CNN) was originally invented to deal with computer vision problems like image recognition but it can achieve remarkable result in NLP tasks such as sentiment analysis [8].

Like the other model for NLP tasks, CNN are build on top of a word embedding layer then it is followed by a convolution layer and a max pooling layer. To explain this, lets first look at The model architecture in the figure 3.10 [23]:

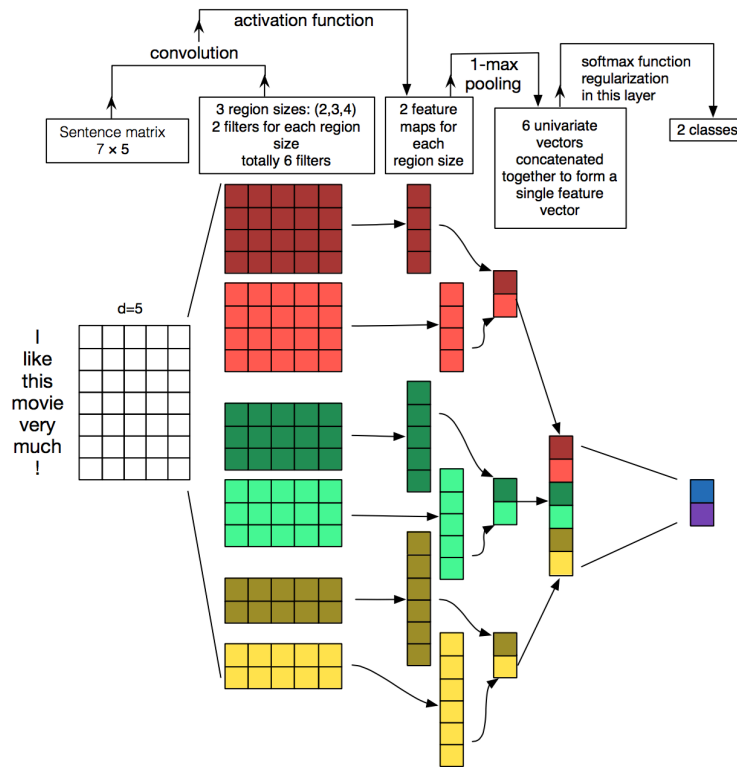


Figure 3.10: Convolutional Network architecture for sentence sentiment

The input (the sentence) will be represented as a matrix where each row represent each word's vector. this way, the width of this matrix is the size of the word embedding vector and the height is the length of the sentence. Now that the matrix is created, a convolution with a filter of a fixed region size is applied.

In NLP, the filter width is always the same size as the word embedding, this makes sense because each time we slide over full rows of the matrix (the words), typically we slide over 2 to 5 words at the same time. For example, applying the first filter with region size of 4 to our input matrix will result in a matrix which height is 4. This resulting Matrix is called a feature map, and because we want to extract as many feature as we can, we will use different region size filters and for each filter we use different coefficient. In the example above we used 3 filters size and for each filter we used 2 different matrices. After the convolution a non-linear activation function is used like ReLU or tanh.

The following step after convolution typically consist of a pooling layer. the

pooling layer subsample the input and then we can see its size reduced. The most common way to do pooling is to apply a max operation to the result of each filter. Like a convolution the pooling have a filter but it usually slides over non overlapping patches from the matrix, taking the max from each patch.

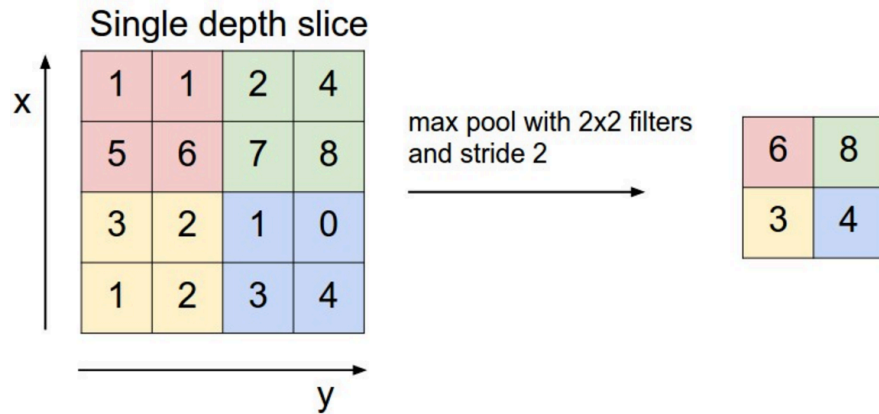


Figure 3.11: A max pooling operation resulting in dimension reduction while conserving important information

We can also apply an average pooling but in practice it is proved that max pooling achieves better results. In CNN, pooling is a main component for two reasons.

- First pooling always provides us with a fixed size output matrix required for classification in the final step regardless the size of the convolution filters.
- Pooling also reduce the output dimensionality and hopefully keeping the most relevant informations. In fact, we have to think of pooling as a specific feature detector such as detecting if the sentence contains a negation like "not amazing" [24].

The final step in this model involves collecting all the outputs from the pooling layer into a single feature vector. This vector will be fed to a Softmax classifier to output in this case a 2 dimension vector. The first case contains the probability that the input sentence is negative and the second one is the probability that the sentence is actually positive.

3.5.4 Neural Network Comparison Summary

Model	Advantages	Limits	Training Time and Accuracy
RNTN	High accuracy in the fine grained sentiment analysis task.	<ul style="list-style-type: none"> • Labelling each word in the sentence is revealed to be a very difficult task. • the RNTN response for a long input is quite slow because the tree parse is quite expensive. 	--
LSTM	--	<ul style="list-style-type: none"> • LSTM only deal with sentences having the same length because we need to initialize the LSTM chain with a fixed number of cells. 	For the configuration used in this project, the training time can take from 6 to 8 hours to reach an accuracy of 79% in the test set.
CNN	Convolution are very fast and they are even implemented on a hardware level on GPUs.	<ul style="list-style-type: none"> • Convolutions and pooling operations lose information about where exactly a feature appeared or not in the sentence but we still capturing and keeping the local information, so the model can still differentiate between not bad, good and bad, not good. If we want our model to capture more global information, then we can increase the filter size. • CNN only deal with sentences having the same length that's why we need to pad all the sentences to a maximum value 	For the configuration used in this project and depending on many hyper-parameter tuning, the training time can take from 2 to 6 hours and can reach an accuracy of 81% in the test set.

Figure 3.12: Neural Network Comparison Summary

The model we decided to use is the convolutional neural network because it is rapid and achieve the state of the art in binary sentence classification with an accuracy of 81%.

3.6 Conclusion

Defining a model for sentence classification is an art. The models may differ in their architecture but they same some common concepts like the word embeddings.

The main problem in this task is dealing with sequences that vary in length that's why we need first to process those sequence and pad them to a maximum length.

Chapter 4

Sentiment Analysis for Steam Games using CNN

4.1 Introduction

In this section, we are going to describe step by step the approach used for sentiment analysis then we are going to briefly describe the production environment and expose some result from our experiment.

4.2 Approach Description

For the current project, we used the CNN model to predict positive/negative sentiment. This is how our network look like generated using tensorflow's tensorboard. We are going to walk-through each layer step by step giving more technical details.

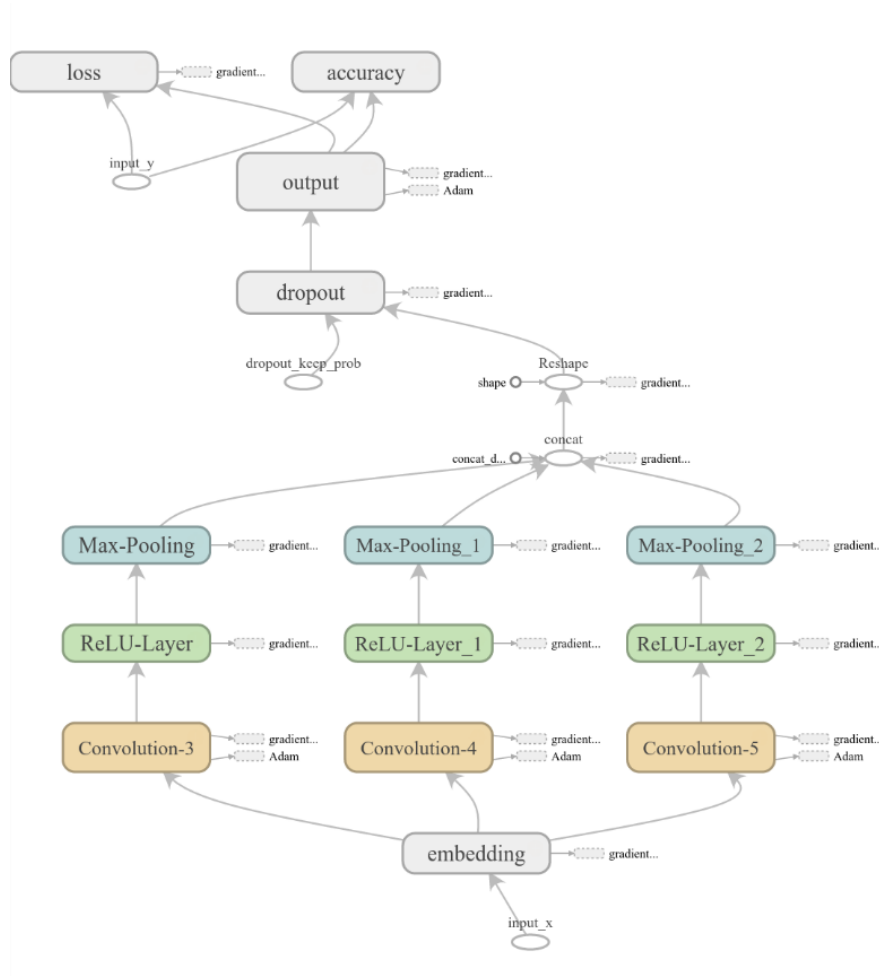


Figure 4.1: The CNN architecture used in our project

4.2.1 Preprocessing

Our Network needs two inputs: the sentences ($input_x$ in the graph) and their labels ($input_y$).

The data-set source used in this project is varied. In fact we extracted some reviews from steam, amazon, yelp and we used the Movie Review data from Rotten Tomatoes. Our data is stored into two raw files. They contains 6831 positive comments and 6831 negative comments and the vocabulary size is over 20000 unique words. our pre-processing step does the following:

1. load the sentences from the date files.
2. Remove any unknown characters from the data.
3. Pad each sentence to a maximum length called $SequenceLength = 59$.

4. build a vocabulary from the unique words in our data and map each one to an integer. For example, we can create a dictionary and order the words by their occurrence. This way each sentence becomes a vector of integers.

For example

The Sentence	The	cat	eats	a	mouse
becomes	1	1721	9744	3	4025

4.2.2 Embedding layer

This is the first layer. It is just a lookup table that maps vocabulary word indices into low-dimensional vector representation.

For better results, We did not learned those vectors from scratch instead we used the pre-trained word2vec from unsupervised learning by mikolov et al [14], trained on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. This way each input now is a matrix of shape $[SequenceLength = 59, EmbeddingSize = 300]$ where each row represents a word.

4.2.3 Convolution Layer

Convolution is applied the same way in image processing except that the *FilterWidth* is always equals to the input width which is the *EmbeddingSize*. This makes sense since our filter is sliding over the rows of the input matrix taking *FilterHeight* words each time. We used three filter sizes, a $[3, 300]$, a $[4, 300]$ and a $[5, 300]$. Also because we want to extract as many features as we can, we created $NumFilters = 128$ of each filter. Applying a filter to the input matrix results in a matrix of shape $[1, n_{out}]$ where $n_{out} = SequenceLength - FilterHeight + 1$. The output matrix is generally called a feature map. This way, we have extracted $3 * 128 = 384$ feature maps.

4.2.4 ReLU layer

This layer is an important layer and it is present in every machine learning architecture because it adds non linearity to the model. We recall that ReLU stands for Rectified Linear unit. It is a pointwise operation defined as:

$$f(x) = \max(x, 0) \quad (4.2.1)$$

In this project, we use the ReLU function thanks to its popularity in machine learning [25]. In fact ReLU is a cost less computation function compared to tanh and the sigmoid functions that involves expensive operation hence

4.2.5 Max pooling

Pooling is a key aspect in CNN models. It is typic to apply it after convolutions. In this project we used max pooling because in practice it gives better results than average pooling. In our task we just take the max value from each feature map and then combine all those values in one big feature vector which size is : $NumFilters * len(FilterSizes) = 384 = S$

4.2.6 The final output

Because we have two classes in this task (negative/postive), the big feature vector from the max pooling operation is multiplied by a matrix wich have the shape $[S = 384, NumClasses = 2]$ obtaining a vector with the shape of $[1, NumClasses]$ called the scores. Each value in this vector correspond to the score for our input sentence to be in a certain class. In order to normalize the scores and obtain probabilities, we can use the Softmax function defined in 2.5.9.

Calculating the loss

The labels $input_y$ are used in this stage to calculate the loss. For this, we used the Cross-Entropy, it has the following formulas:

$$CrossEntropy = - \sum_{i=1}^n (y_i \log \hat{y}_i) \quad (4.2.2)$$

Remember that in the section 2.5.8, we said that the loss is not calculated for a single \hat{y} but for $n = BatchSize$.

4.2.7 Calculating the accuracy

The accuracy is computed to see if our model is performing well in the test set. Depending on many hyper-parameters tuning such as *NumFilter*, *FilterSize* ..., the model can reach 81% accuracy on the test set.

4.2.8 Overfitting

you may have noticed that we didn't talk about the Dropout component in the network graph shown in figure 4.1. Overfitting is a common big problem in machine learning and there is no exception in our case. In fact, after many steps, the loss in our training example continue to decrease while the loss has started to increase dramatically in the test set.

Overfitting is what happens when a model fits itself to "noise", not signal. Intuitively, overfitting means that the model has perfectly adjusted its parameters to match exactly and only the training set. For example if the model has been over-trained on the sentences "I like eating chocolate" and "I like playing this game" as positive sentences it may predict that the sentence "I like watching movies" as negative because it learned that only the sentences containing the words "like", "chocolat" and 'game' are positive and nothing else. Hopefully there is a couple of methods to reduce overfitting [26].

- Dropout: Dropout is the most popular method to regularize neural networks. The idea is very simple. Because we don't want our model to be over-trained on the same sentences, we eliminate some of them at each training step with a probability p . Using a grid search, the best value of p is proved to be 0.5.
- Max norm constraints for the weights: another form of regularization is to enforce an absolute upper bound on the magnitude of the weight vector for every neuron. In practice, this corresponds to performing the parameter update as normal, and then enforcing the constraint by clamping the weight matrix W

of every neuron to satisfy $\|W\|_2 < c$. In the present work $c = 3$.

- L2 regularization: It can be implemented by penalizing the squared magnitude of all parameters directly in the loss. That is, for every weight W in the network, we add the term λW^2 to the objective, where λ is the regularization strength. We used $\lambda = 0.2$. The L2 regularization has the intuitive interpretation of heavily penalizing peaky weights and preferring diffuse weights.

4.3 Implementation

4.3.1 Production environment

Optiplex-360

- Central Processing Unit: Intel Dual Core
- Graphics Processing Unit: Nvidia
- Memory: 3Gb
- Operating System: Ubuntu 14.04 LTS

Server

- Central Processing Unit: Intel Xeon E504
- Graphics Processing Unit: Nvidia
- Memory: 16Gb
- Operating System: Ubuntu 14.04 LTS

4.3.2 Languages and Frameworks

Python

python is a widely used high level interpreted programming language. Python supports multiple programming paradigms, including object oriented, imperative and procedural styles [27].



python is a cross-platform language and it's major implementations are Cpython (implemented with C) IronPython (implemented in C#) and Jython (implemented in Java). The most used is the Cpython called just python or pure python.

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object.
- sophisticated (broadcasting) functions.
- tools for integrating C/C++ and FORTRAN code.
- useful linear algebra, Fourier transform, and random number capabilities.

Besides its obvious scientific uses, NumPy can also be used as an efficient multidimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. Numpy is licensed under the BSD license, enabling reuse with few restrictions [28].



Tensor-flow



TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API [29].

TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

TensorFlow comes with an easy to use Python interface and a no-nonsense C++ interface to build and execute your computational graphs. Tensorflow comes with tensorboard which allows us to visualize the data flow graphs.

Bottle framework



Bottle is a fast, simple and lightweight WSGI micro web-framework for python. It is distributed as a single file module and has no dependencies other than the python standard library.

A micro framework means that it only implements the basic concept for a web server, for example there is no build in support for session, that's why we have to use other extension or implement them ourselves.

Beaker sessions

Beaker is a library for caching and sessions for use with web applications and standalone Python scripts and applications. It comes with WSGI middleware for easy drop-in use with WSGI based web applications, and caching decorators for ease of use with any Python based application such as bottle web application.

Gevent server

gevent is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libev event loop [30]. Gevent allow us to write multithreaded bottle web application.

Bootstrap

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.



jQuery

jQuery is a fast, small, and feature-rich JavaScript library.

It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of

browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.



Plotly



Plotly is a framework for plotting graphs, charts and visualizing data in both 2D and 3D. It is compatible with multiple of tools such as matlab, javascript, excel as well Python. Plotly outputs the graphs in a HTML file to visualize in the browser [31].

4.3.3 Experiment and Result Analysis

The figures 4.2 and 4.3 respectively represent the accuracy and the loss in our first attempt to train the CNN after $NumEpochs = 220$ or 30000 steps.

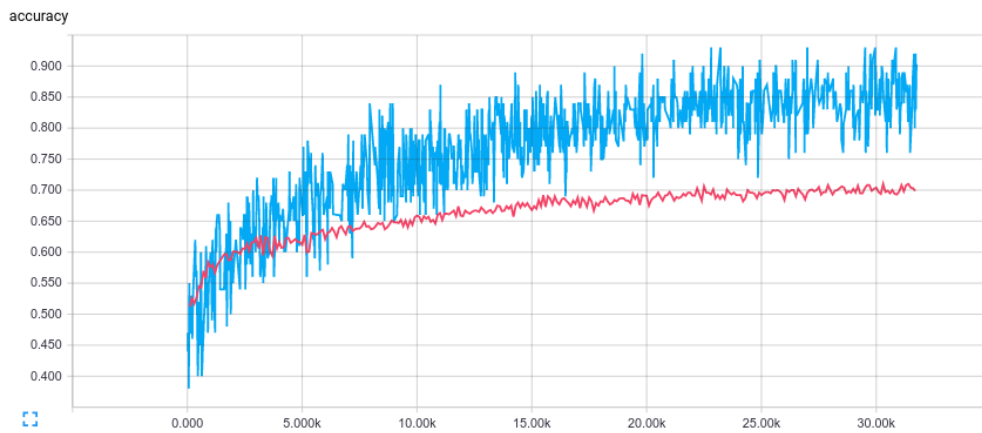


Figure 4.2: Accuracy for the training set (blue plot) and the test set (red plot)

In each step, the model computes the accuracy for a $batchSize = 100$ of data, when the model iterates through all the generated batch from the data, it terminates an epoch. We can see that the blue training curve is a bit agitated because we use small batch sizes. If we use some larger batches or even the whole training set, we can get a smoother blue curve. Even disturbed, we can see that the accuracy is always increasing, the model is trying to learn.

Because test accuracy is significantly below training accuracy it seems like our network is overfitting the training data. In the figure 4.3, after the step $s_0 = 2000$, the loss in the test set (the red plot) has begin to increase while the loss in the training example is decreasing as usual.

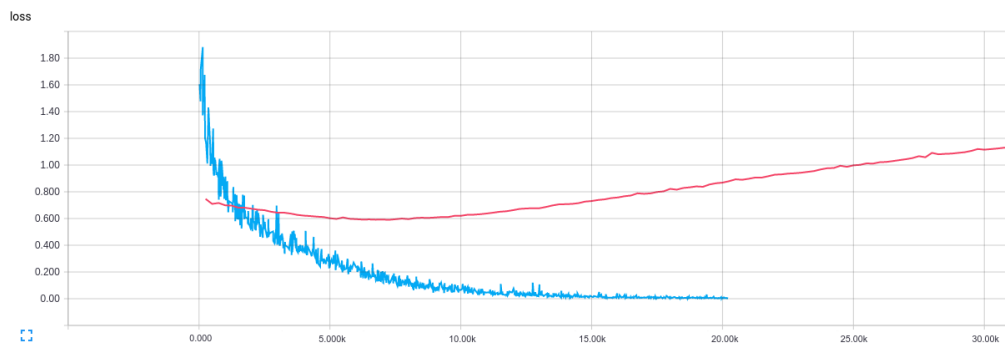


Figure 4.3: The model overfitting the data

Applying all the methods to reduce overfitting described in the section 4.2.8 results in a better shape for the loss curve, even after so many steps overfitting is

greatly reduced.

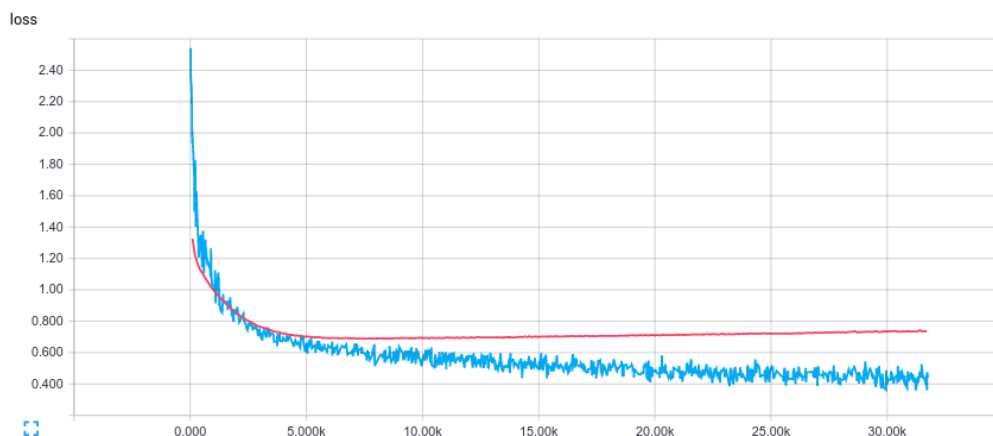


Figure 4.4: The loss curve after applying regularization methods

Also the accuracy for the test set is increased to reach 76%.

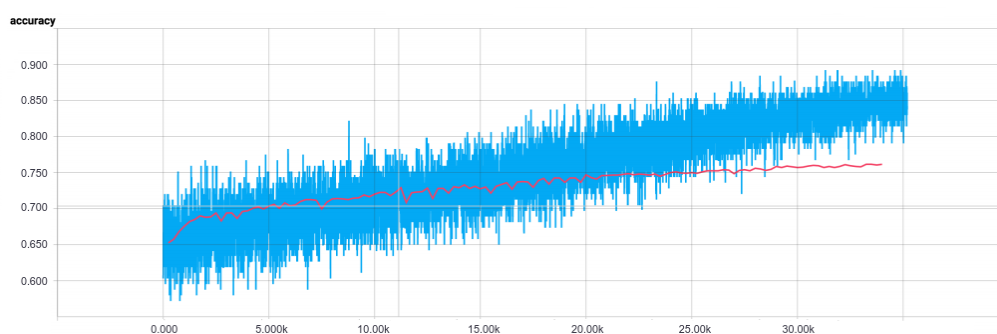


Figure 4.5: The accuracy curve after applying regularization methods

This accuracy is also achieved thanks to the use of the pre-trained word2vec.

4.3.4 Application

After the training our learned model parameters are saved to a local file and can be used in production environment. This is the main interface in our web application. We can choose the sentiment predict service or the sentiment plot.

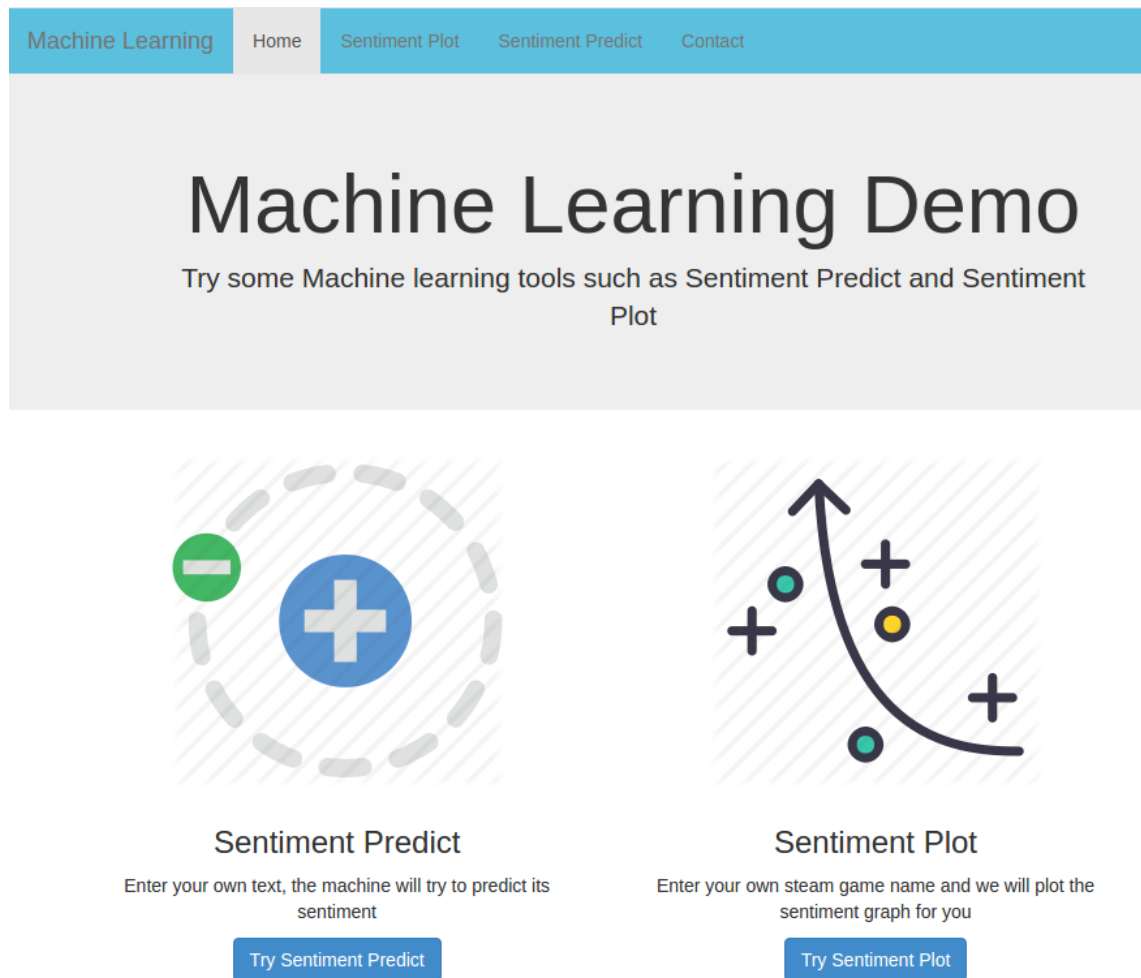
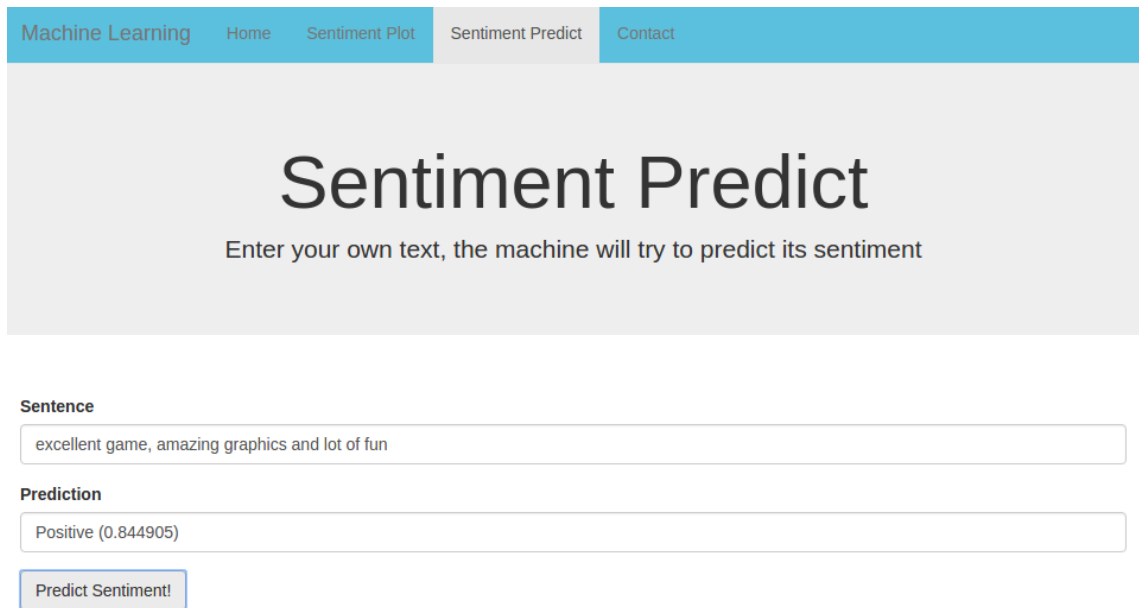


Figure 4.6: Home page

Sentiment predict

In this page, you can enter your own text, the model will analyse the text and predict if it is actually positive or negative



The screenshot shows a web application with a blue header bar containing navigation links: "Machine Learning", "Home", "Sentiment Plot", "Sentiment Predict" (which is highlighted), and "Contact". Below the header, the main content area has a light gray background. It features the title "Sentiment Predict" in a large, bold, black font, followed by the instruction "Enter your own text, the machine will try to predict its sentiment". Below this, there is a form with two input fields. The first field is labeled "Sentence" and contains the text "excellent game, amazing graphics and lot of fun". The second field is labeled "Prediction" and contains the text "Positive (0.844905)". At the bottom of the form is a button labeled "Predict Sentiment!".

Figure 4.7: The model accurately predicting the positive sentiment

Some other sample example:

The trailer was amazing but at the end the game was not that great

Negative (0.668130)

Terrible gameplay, terrible graphic why would someone buy this game?

Negative (0.941005)

I enjoyed playing this game so much, innovative game-play and nice community.

Positive (0.736044)

I didn't expect that this game would be so good

Negative (0.803032)

Sentiment plot

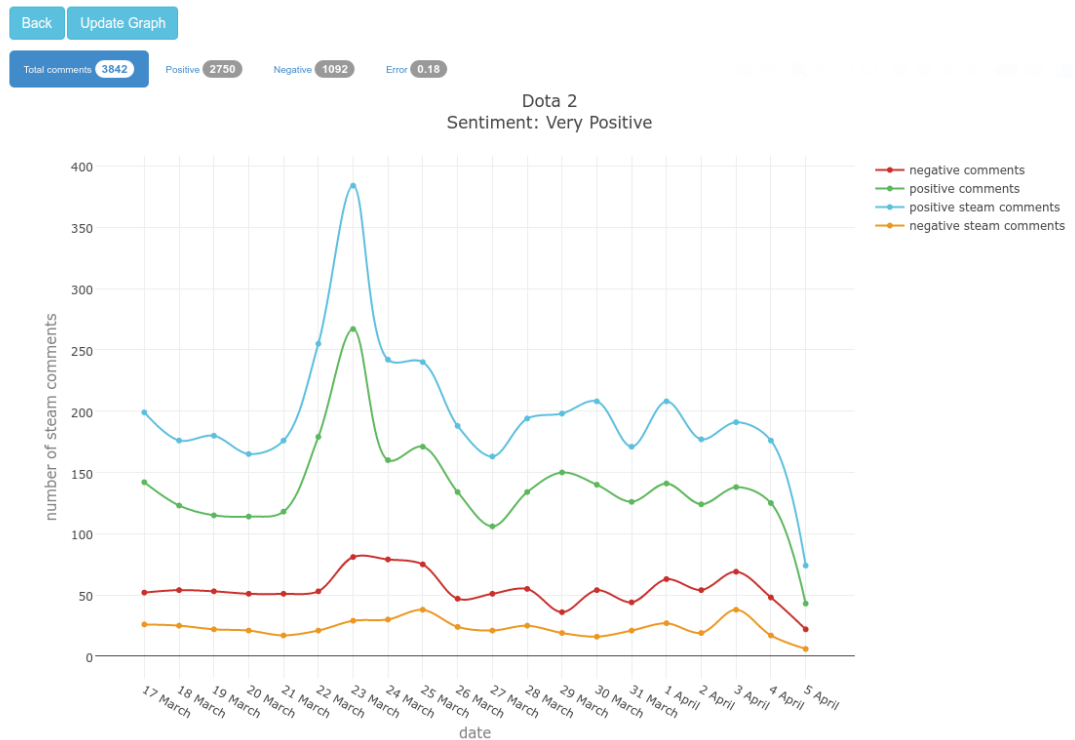


Figure 4.8: the model predicted a very positive sentiment for the game Dota 2 with an accuracy of 82%.

The green plot is the predicted positive sentiment, the red curve is the negative predicted sentiment. For test only we decided to also include the correct labels from steam, the blue curve is the correct positive curve and the orange one is actually the correct negative curve. We can see that our model successfully analysed the sentiment for 3842 comments over the last twenty days.

Some minor error can occur because there is some "noisy" comments e.g in some cases people can say "I hate this game" while they meant the inverse. The model fails to predict a positive sentiment for those comments. This is some "noisy" comments extracted from steam for the Dota 2 game.



Figure 4.9: A noisy positive comment



Figure 4.10: Another noisy positive comment

The peak in the green curve can not be unnoticed especially when we are analysing the sentiment. In fact, we notice a surge in the positive comment between 22 and 24 march. This can be explained by the release of a new version of the game with many improvement and bug fixes.

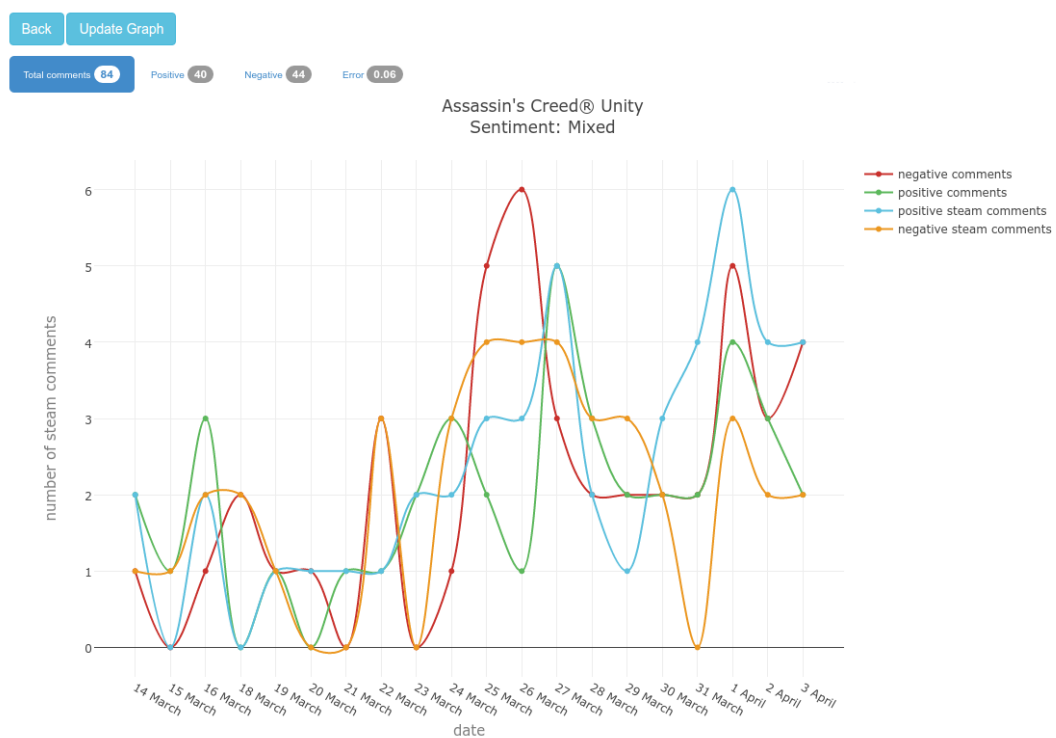


Figure 4.11: The model accurately predicting a mixed sentiment for the AAA game Assassin's Creed Unity with an accuracy of 94%.

As we can see from the graph the community wasn't happy with the latest assassin's creed episode and the curves are very disturbed.

4.4 Conclusion

Our implemented solution of the CNN model achieved excellent result in predicting the sentiment in the steam games reviews.

Conclusion and perspectives

This project has required both theoretical and practical skills. The first step was to understand the machine and deep learning concepts and the second was to properly apply them. The patience was greatly rewarding and we successfully implemented a CNN that achieves the desired accuracy.

The solution provided in the current work can be improved in several ways, we enhance our prediction with a fine grained sentiment analysis like predicting a very positive sentiment or a very negative one.

Also we can introduce a model for product feature extraction. This means that the system will be able to predict a sentiment and tells us the reasons behind this sentiments, for example the system can predict that this sentence "I like this game because the game-play is funny and graphics are amazing" is positive and the user is happy with the game-play and the graphics.

References

- [1] Stanford. Neural networks, 2013. URL http://ufldl.stanford.edu/wiki/index.php/Neural_Networks.
- [2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, pages 303–314, 1989.
- [3] Huaiqin Wu. Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions. *Information Sciences 179.19*, pages 3432–3441, 2009.
- [4] Christopher Olah. Understanding lstm networks, 2015. URL <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [5] Andrej Karpathy. Convolutional neural network for visual recognition, 2014. URL <http://cs231n.github.io/convolutional-networks/>.
- [6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *In Proceedings of Neural Information Processing Systems (NIPS) 2012*, page 1, 2012.
- [7] aaron vandoord, sander dieleman, and benjamin schrauwen. Deep content-based music recommendation. *In Proceedings of ACL 2005*, page 26432651, 2013.
- [8] Yoon Kim. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

- [9] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Berlin Heidelberg, 2012.
- [10] Michael Nielsen. How the backpropagation algorithm works, 2016. URL <http://neuralnetworksanddeeplearning.com/chap2.html>.
- [11] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *In Proceedings of ICASSP 2013*, page 1, 2013.
- [12] Yoshua Bengio, Rjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3: 1137–1155, 2003.
- [13] Wen tau Yih, Kristina Toutanova, John Platt, and Chris Meek. Learning discriminative projections for text similarity measures. *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, 2011. URL <http://research.microsoft.com/apps/pubs/default.aspx?id=150018>.
- [14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *In Proceedings of NIPS 2013*, 2013.
- [15] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.
- [16] Google Inc. Vector representations of words, 2015. URL <https://www.tensorflow.org/versions/r0.8/tutorials/word2vec/index.html>.
- [17] Chris McCormick. Word2vec tutorial - the skip-gram model, 2016. URL <http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model>.
- [18] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research* 9, pages 2579–2605, 2008. URL https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf.

- [19] Christopher Olah. Visualizing representations: Deep learning and human beings, 2015. URL <http://colah.github.io/posts/2015-01-Visualizing-Representations/>.
- [20] Christopher Olah. Deep learning, nlp, and representations, 2014. URL <http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/>.
- [21] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *Conference on Empirical Methods in Natural Language Processing (EMNLP 2013, Oral)*, page 1, 2013.
- [22] Kyunghyun Cho and Pierre Luc Carrier. Lstm networks for sentiment analysis, 2016. URL <http://deeplearning.net/tutorial/lstm.html>.
- [23] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification. *Computing Research Repository (CoRR)*, 2015. URL <http://arxiv.org/abs/1510.03820>.
- [24] Denny Britz. Understanding convolutional neural networks for nlp, 2015. URL <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>.
- [25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012. URL <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolut>
- [26] Andrej Karpathy. Convolutional neural network for visual recognition, 2016. URL <http://cs231n.github.io/neural-networks-2/#reg>.
- [27] Guido van Rossum. Python, 2015. URL <https://www.python.org/>.
- [28] Travis Oliphant. Numpy, 2006. URL <http://www.numpy.org/>.
- [29] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, San-

- jay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <http://tensorflow.org/>. Software available from tensorflow.org.
- [30] Denis Bilenko. Gevent, 2010. URL <http://www.gevent.org/>.
- [31] Alex Johnson, Jorge Barcroft, and Potato Reese. Plotly, 2012. URL <https://plot.ly/>.

ملخص

العمل الحالي لنهاية المشروع العام هو حول استخراج المشاعر من تعليقات حول الألعاب في موقع Steam. ان تطبيق نهج ساذج لهذه المهمة لا يعطينا النتيجة المتوقعة لهذا السبب لجأنا إلى أساليب التعلم العميقة وبشكل أكثر تحديدا إلى الشبكات العصبية CNN. تم بناء هذه الشبكة على word2vec. هو نوع من التعلم غير المراقب و تدرب بشكل مستقل من شبكتنا. وهو يتألف من تحويل الكلمات إلى جدول من الأعداد. هذه التجربة مع الشبكة العصبية أدى بنا إلى تحقيق نتيجة ممتازة تصل في المتوسط إلى 83% من الدقة. الكلمات المفتاحية: word2vec، الشبكة العصبية CNN، استخراج المشاعر، موقع Steam

Abstract

The current work for the end of year project is about extracting the sentiment from games comments in the steam platform.

Applying a Naive approach for this task would not give us the result expected that's why we turned to deep learning methods and more specifically to convolutional neural networks CNN.

This network is built on top of word embedding word2vec; it is a kind of unsupervised learning that it trained independently from our network. It consists of transforming words into numerical vector.

Our experiment with the convolutional neural network overcome the limit of the naive approach and led us to achieve excellent result reaching an average of 83\% accuracy.

Keywords: word2vec, convolutional network CNN, Sentiment Analysis, Steam platform.

Resumé

Ce projet de fin d'étude consiste à extraire les sentiments des revues de jeux dans la plateforme Steam.

Les méthodes classiques pour cette tache sont très limites, nous avons utilisé un réseau de neurones convolutionnel CNN.

Ce réseau est rajouté sur une couche d'apprentissage unsupervisé appelé word2v. Cet apprentissage consiste à transformer un mot en un vecteur numérique.

Nos essaie avec CNN a conduit à des résultats excellent pour atteindre une précision de 83% en moyenne.

Mot-clé : word2vec, réseau convolutionnel CNN, analyse des sentiments, plateforme Steam.