Deep Sentiment Analysis on Tumblr

Anthony Hu

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Applied Statistics

Department of Statistics
University of Oxford
Oxford, United Kingdom

September 2017

Declaration

The work in this thesis is based on research carried out at the Department of Statistics, University of Oxford. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

Many thanks to my supervisor Seth Flaxman and to Kellogg College.

Deep Sentiment Analysis on Tumblr

Anthony Hu

Submitted for the degree of Master of Science in Applied Statistics
September 2017

Abstract

This thesis proposes a novel approach to sentiment analysis using deep neural networks on both image and text.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Tumblr data

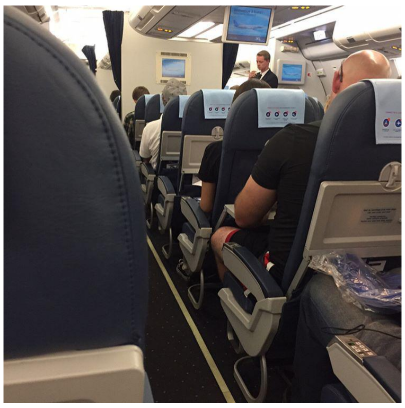
2.1 Overview of the data

Tumblr's posts were extracted using the official API thanks to their tags that were taken as the ground truth. The tags represent the user's emotion: happy, sad, angry, surprised, scared or disgusted. The data extraction took several weeks due to the API's limitations: 1,000 requests per hour and 5,000 requests per day, with each request containing 20 posts. The final dataset has about one million posts and six different emotions.

Need to talk about preprocessing non-english posts

Here are examples of posts with their associated emotions:

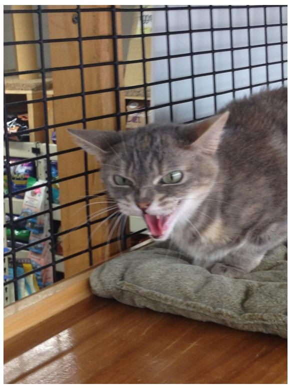
(a) **Happy**: “Just relax with this amazing view #bigsur #california #roadtrip #usa #life #fitness (at McWay Falls)”



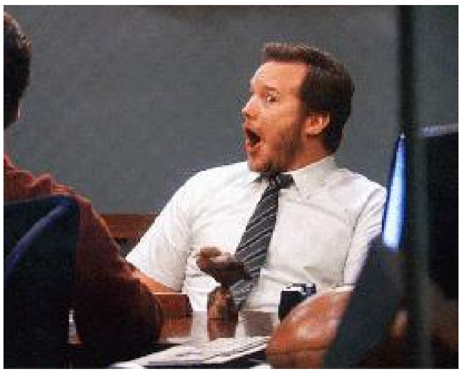
(b) **Scared**: “On a plane guys! We’re about to head out into the sky to Paris, France #Paris #trip #kinda #nervous #fun #vacations”

It's okay
even the sky cries sometimes

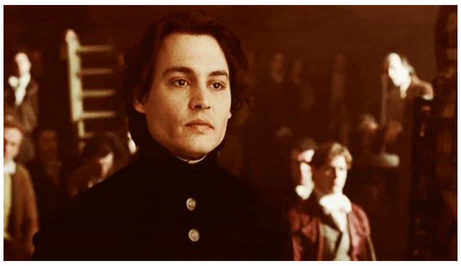
(c) **Sad**: “It’s okay to be upset. It’s okay to not always be happy. It’s okay to cry. Never hide your emotions in fear of upsetting others or of being a bother If you think no one will listen. Then I will.”



(d) **Angry**: “Tensions were high this Caturday...”



(e) **Surprised**: “Which Tea? Peppermint tea: What is your favorite gif right now?”



(f) **Disgusted**: “Me when I see a couple expressing their affection in physical ways in public”

Figure 2.1: Some examples of Tumblr posts [1]

Chapter 3

Visual recognition

The pictures accompanying the posts usually help a lot in determining the emotion of the user. For instance, happy photos might contain landscapes with the sun or a beach while sad ones might have really dark colors. To analyse the images, we'll use convolutional neural networks, which achieve state-of-the-art performances in many visual recognition tasks. First we'll explain how they work and then we'll dive into the architecture we've used for deep sentiment analysis.

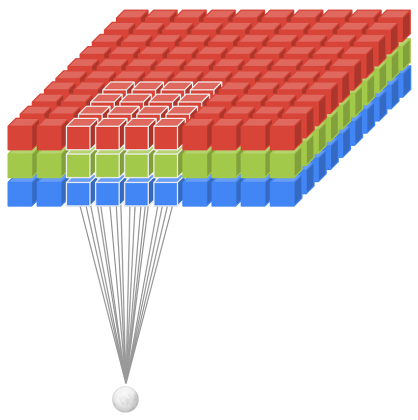
3.1 Convolutional neural networks

A convolutional neural networks, often called ConvNets, can be seen as a simulation of the human visual cortex, that is to say an aggregation of plenty of receptive fields. (some illustrations might be helpful here)

3.1.1 Convolutional layer

Take an image of dimension $(h, w, 3)$ with h the height, w the width and 3 representing the number of channels (red, blue, green). If you simply flatten that image and transform it into a vector of size $h \times h \times 3$ and feed that vector to a neural network, you'll get not-so-good results as you've thrown away all the spatial information. Convolutions extract that spatial information and work the following way:

- Each convolution is described by a filter F of size $(f, f, 3)$, f usually being in the range $[1, 13]$.


(a) One operation of convolution

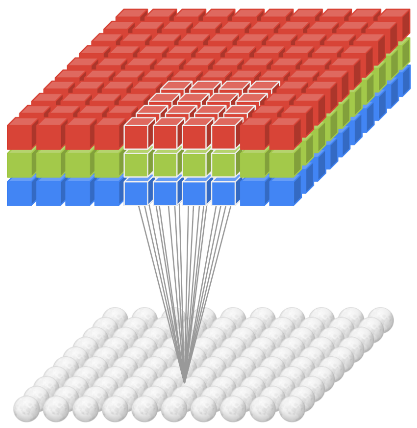

(b) The result of the convolution

Figure 3.1: A convolution, each neuron been a 'receptive field' [2]

- Position the filter on the upper left of the image and element-wise multiply with the filter, then sum those numbers to obtain a raw number.
- Slide across the image, one pixel at a time horizontally and vertically, and repeat the previous operation.

By sliding through the image, you would get a new matrix of dimension $(h_{new}, w_{new}, 1)$, with $h_{new} = h - f + 1$ and $w_{new} = w - f + 1$. However, we usually don't want to reduce the size of our input image that fast, as we usually have several convolutions. To ensure that the image has the same size, zero-padding is used: we add p zeros to the borders of the input image to preserve the spatial size of the input. With zero-padding, h_{new} becomes: $h_{new} = h + 2p - f + 1$, and we want that h_{new} equals h:

$$h + 2p - f + 1 = h \tag{3.1}$$

Therefore, $p = \frac{f-1}{2}$.

A convolution extract information about the image such as edges or blotches of some color (Figure 3.2). The grayscale and edges filters were hardcoded but in a ConvNet setting, the weights of the filter F are learned through optimising a loss function – in our case, a metric measuring how accurate our predictions of the emotions are. The network will learn weights that will detect features that will be most relevant to our specific task. Most of the time, more than a single convolution are used: just repeat that operation d (the depth) times, to create a new tensor

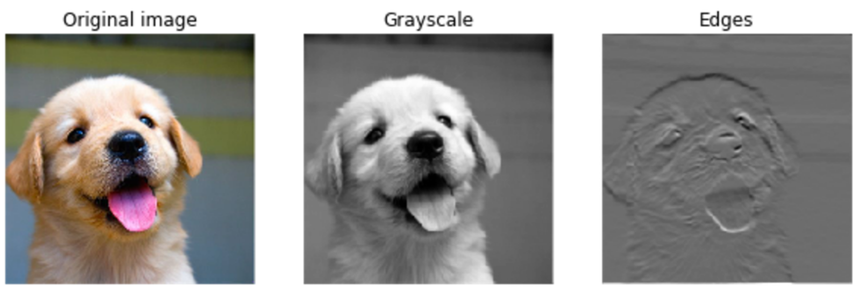


Figure 3.2: Examples of convolution

of dimension (h, w, d). We can then apply convolutions on that new image. First layers will recognise simple features such as edges or aggregation of colors, and deeper layers might activate on faces, wheels and so on.

3.1.2 ReLU layer

Stacking convolutions is nice, but as it is, we are only creature features that are linearly dependent of the input pixels: we could replace all the convolutions with a single matrix multiplication. In order to learn more interesting functions, we have to add non-linearities. Historically, the popular choice was the sigmoid function defined as:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

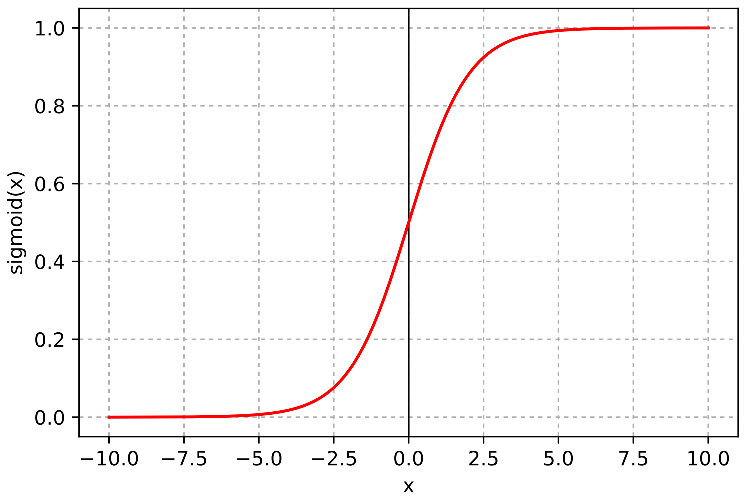


Figure 3.3: Sigmoid function

The sigmoid function is the simplest function having values between 0 and 1 mimicking the biological neurons 'firing' to their inputs. However, when the network is learning to minimise a loss function through backpropagation, the gradients tend to vanish to zero as the sigmoid's derivative goes to zero for high negative and positive values. The most popular choice is now the Rectified Linear Unit (ReLU) defined as:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.3}$$

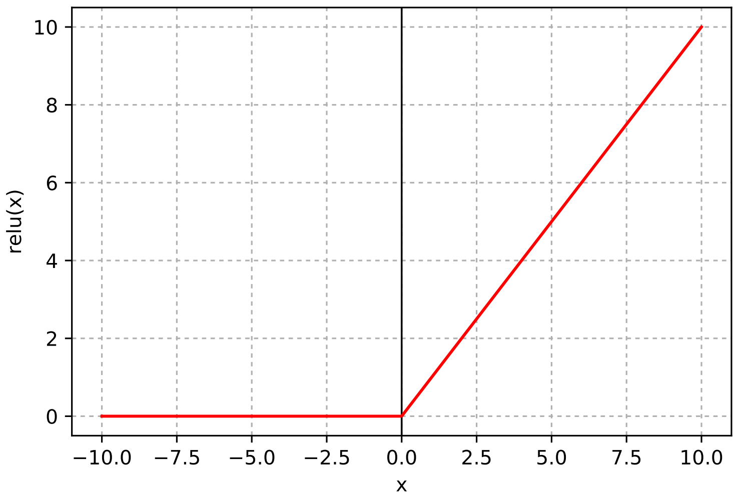


Figure 3.4: Sigmoid function

The ReLU's gradient is non-saturating for highly excited neurons which turns out to be a nice property to learn faster. In the network, each layer of convolution is followed by a ReLU layer, that simply applies the function $\max(0, x)$ to each neuron.

3.1.3 Pooling layer

After a few iterations of convolutions, inserting pooling layers in-between convolutional layers might be a good idea to control the spatial complexity of the network. More concretely, what is most used in practice is the 2x2 max-pooling:

- Pick a channel among the d ones.
- Position yourself on the top-left 2x2 square of the image and take the max.

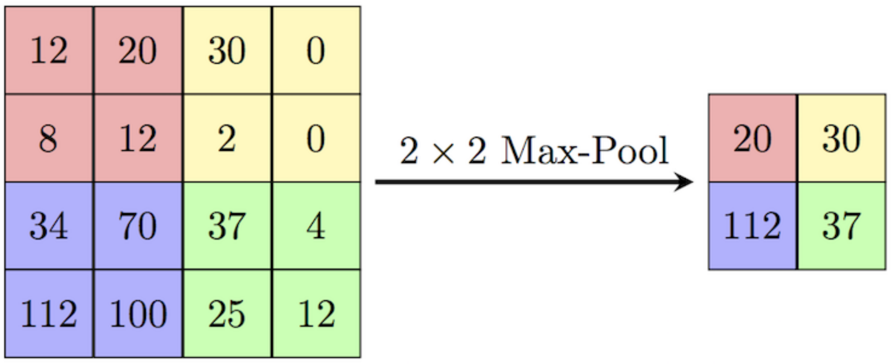


Figure 3.5: Max pooling [3]

- Repeat by sliding through the image vertically and horizontally with a stride (step) of 2.

After applying max pooling to each channel, the resulting image's dimension is $(\frac{h}{2}, \frac{w}{2}, d)$ and we have discarded 75% of the neurons (as in each max-pool operation, we only keep the maximum neuron among the fours), effectively reducing the number of parameters and controlling overfitting. Pooling is also useful to create translation-invariant features.

3.1.4 An example of ConvNet

Here is an example of a convolutional neural network with an input image of size $(100, 100, 3)$:

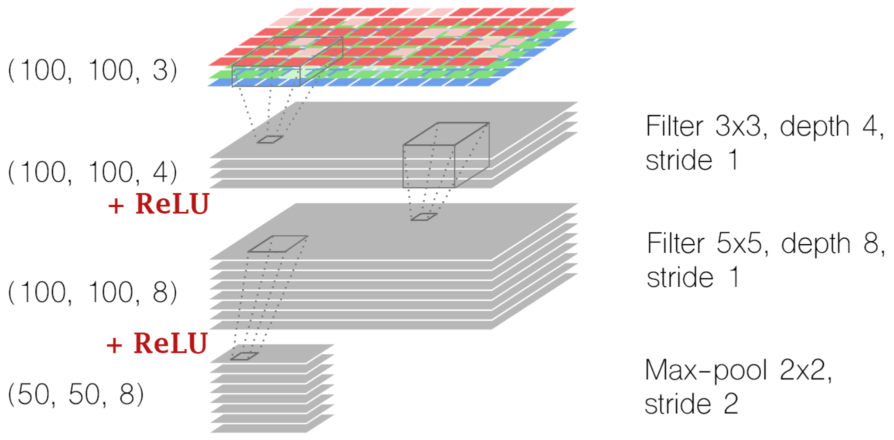


Figure 3.6: An architecture of a neural network [2]

- A first convolution with a filter of size 3×3 is applied, with depth 4, stride 1 and zero-padding of 1.
- A ReLU layer.
- A second convolution with a filter of size 5×5 is applied, with depth 8, stride 1 and zero-padding of 2.
- A ReLU layer.
- Max-pooling of size 2×2 with stride 2, reducing the height and width by 2.

After the last operation, the neurons are reshaped into a vector that can be fed to the traditional fully-connected layers of neural networks.

3.1.5 Deep convolutional networks

Best results are achieved using deep convolutional networks, that is to say by stacking many layers of convolutions, ReLU, max-pool. But what exactly is 'many'? Let us have a look at the main Computer Vision competition: ImageNet Large Scale Visual Recognition (ILSVR).

1. **AlexNet** [4]: The first popular convolutional network, developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton, that outperformed the other competitors at the ILSVR 2012 by a large margin: top-5 error of 16% compared to the runner-up with 26%. AlexNet has 5 convolutional layers (followed by ReLU), 3 max-pool layers and 3 fully-connected layers, producing a 8-layer deep network (not counting the max-pooling as it doesn't have any parameters).
2. **GoogLeNet** (also known as Inception) [5]: This is the winner of the ILSVR 2014 with a top-5 error of 6.7% . This architecture used the 'Inception Module' which allowed to drastically reduce the number of parameters: from 60M for AlexNet to 4M for GoogLeNet.

Chapter 4

Natural Language Processing

Text analysis

4.1 Section 1

Chapter 5

Recurrent Neural Networks for text generation

5.1 Section 1

Chapter 6

Useful maths in LaTeX

6.1 Equations related

$$\frac{\partial u_1}{\partial t} = \Delta w_1 \quad \text{in } \Omega, t > 0, \tag{6.1}$$

$$\frac{\partial u_2}{\partial t} = \Delta w_2 \quad \text{in } \Omega, t > 0, \tag{6.2}$$

where

$$w_1 = \frac{\delta F(u_1, u_2)}{\delta u_1}, \tag{6.3}$$

$$w_2 = \frac{\delta F(u_1, u_2)}{\delta u_2}, \tag{6.4}$$

$$\begin{aligned} F(u_1, u_2) &= b_1 u_1^4 - a_1 u_1^2 + c_1 |\nabla u_1|^2 \\ &\quad + b_2 u_2^4 - a_2 u_2^2 + c_2 |\nabla u_2|^2 \\ &\quad + D \left(u_1 + \sqrt{\frac{a_1}{2b_1}} \right)^2 \left(u_2 + \sqrt{\frac{a_2}{2b_2}} \right)^2 . \end{aligned} \tag{6.5}$$

$$U_1^n = \sum_{i=1}^{J} U_{1,i}^n \eta_i, \quad W_1^n = \sum_{i=1}^{J} W_{1,i}^n \eta_i, \tag{6.6}$$

$$U_2^n = \sum_{i=1}^{J} U_{2,i}^n \eta_i, \quad W_2^n = \sum_{i=1}^{J} W_{2,i}^n \eta_i, \tag{6.7}$$

We also use the following notation, for $1 \leq q < \infty$,

$$L^q(0,T;W^{m,p}(\Omega)) := \left\{\eta(x,t) :\ \eta(\cdot,t) \in W^{m,p}(\Omega), \int_0^T \|\eta(\cdot,t)\|_{m,p}^q \, dt < \infty\right\},$$
$$L^\infty(0,T;W^{m,p}(\Omega)) := \left\{\eta(x,t) : \eta(\cdot,t) \in W^{m,p}(\Omega),\ \operatorname*{ess\,sup}_{t\in(0,T)}\|\eta(\cdot,t)\|_{m,p} < \infty\right\},$$

Cases

$$|v|_{0,r} \leq C|v|_{0,p}^{1-\mu}\|v\|_{m,p}^{\mu}, \qquad \text{holds for } r \in \begin{cases} [p,\infty] & \text{if } m-\frac{d}{p} > 0, \\ [p,\infty) & \text{if } m-\frac{d}{p} = 0, \\ [p, -\frac{d}{m-d/p}] & \text{if } m-\frac{d}{p} < 0. \end{cases} \tag{6.8}$$

6.2 Writing

Lemma 6.2.1 Let $u, v, \eta \in H^1(\Omega)$, $f = u - v$, $g = u^m v^{n-m}$, $m, n = 0, 1, 2$, and $n - m \geq 0$. Then for $d = 1, 2, 3$,

$$\left|\int_\Omega fg\eta dx\right| \leq C|u-v|_0\ \|u\|_1^m\ \|v\|_1^{n-m}\ \|\eta\|_1. \tag{6.9}$$

Proof: Note that using the Cauchy-Schwarz inequality we have

$$|(u)^m v^{n-m}|_{0,p} \leq \begin{cases} |u|_{0,2mp}^m\ |v|_{0,2(n-m)p}^{(n-m)} & \text{for } n-m \neq 0, \text{ and } m \neq 0, \\ |u|_{0,mp}^m \text{ or } |v|_{0,(n-m)p}^{(n-m)} & \text{for } m = 0, \text{ or } n-m = 0 \text{ respectively.} \end{cases}$$

Noting the generalise Hölder inequality and the result above we have

$$\begin{aligned} \left|\int_\Omega fg\eta dx\right| &\leq |u-v|_0\ |u^m v^{n-m}|_{0,3}\ |\eta|_{0,6}, \\ &\leq |u-v|_0\ \ |\eta|_{0,6} \begin{cases} |u|_{0,6}^2 & \text{for } m = 2, \\ |u|_{0,6}\ |v|_{0,6} & \text{for } m = 1, \\ |v|_{0,6}^2 & \text{for } m = 0, \end{cases} \\ &\leq C|u-v|_0\ \|u\|_1^m\ \|v\|_1^{n-m}\ \|\eta\|_1, \end{aligned}$$

where we have noted (6.8) to obtain the last inequality. This ends the proof. $\square$

We consider the problem:

(P) Find $\{u_i, w_i\}$ such that $u_i \in H^1(0,T;(H^1(\Omega))') \cap L^\infty(0,T;H^1(\Omega))$ for *a.e.* $t \in (0,T)$, $w_i \in L^2(0,T;H^1(\Omega))$

$$\left\langle \frac{\partial u_1}{\partial t}, \eta \right\rangle$$

Chapter 7

Conclusions

Bibliography

[1] **Tumblr photos:**
http://fordosjulius.tumblr.com/post/161996729297/just-relax-with-amazing-view-ocean-and
http://ybacony.tumblr.com/post/161878010606/on-a-plane-bitchessss-we-about-to-head-out
https://little-sleepingkitten.tumblr.com/post/161996340361/its-okay-to-be-upset-its-okay-to-not-always-be
http://shydragon327.tumblr.com/post/161929701863/tensions-were-high-this-caturday
https://beardytheshank.tumblr.com/post/161087141680/which-tea-peppermint-tea-what-is-your-favorite
https://idreamtofflying.tumblr.com/post/161651437343/me-when-i-see-a-couple-expressing-their-affection

[2] **Convolution images:** M. Gorner, Tensorflow and Deep Learning without a PhD, Presentation at *Google Cloud Next '17*
https://docs.google.com/presentation/d/1TVixw6ItiZ8igjp6U17tcgoFrLSaHWQmMOwjlgQY9co/pub?slide=id.g1245051c73_0_2184
The slide on the convolutional neural network was adapted to our architecture.

[3] **Max pooling:** Cambridge Spark
https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

[4] A. Krizhevsky, I. Sutskever and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

[5] C. Szegedy et al., Going deeper with convolutions. In *CVPR*, 2015.

Appendix A

Basic and Auxiliary Results

A.1 Basic Results