Neural Networks and its Applications

Orientador: Prof. Dr. João Batista Florindo - IMECC/UNICAMP
Aluno: Gustavo L. Machado - RA: 169356 - IMECC/UNICAMP

2^{th} Semester of 2018

1 Introduction

Text recognition and interpretation is essential to our daily lives. The identification and recognition of texts, either printed, typed or handwritten can be used to automate many different tasks.

Examples of application are autonomous car, which need to be able to interpret road signs, machines that can scan documents and detect signatures or read a letter and output a text file, and much more.

Those problems are deeply studied by the computer vision community, and deep learning is extensively used to seek solutions for them.

On this report we will explain what is an Artificial Neural Network. Focusing on two of its variation, Convolutional Neural Networks and Recurrent Neural Networks and how they perform in text recognition and interpretation.

2 Artificial Neural Networks

Artificial Neural Networks (ANN) are a type of programming that is based on the way that the human brain works. They were designed so that they could learn by viewing examples, like humans do.

With an algorithm that can learn with examples the programmer does not have to think on every possible situation that the problem he wants to solve may have, being able to reduce the number of *if-else* in the code. This makes the code simpler and much more robust, in the sense that the same code can be used to solve more than one problem and more general problems. Deep down, neural networks are basically functions applied to a variable. But how could something that sounds that simple be as powerful as said above?

To better understand it let's look at an ANN structure: Basically neural networks are made of neurons, weights and biases, see Figure 1. The neurons are the functions, the weights are constants that multiply the variables used by the function and the biases are a constant that is used as a threshold. By that logic an input is given to the network, then weights are applied to them and finally the input is used in the function, the neuron activates, and an output is given.

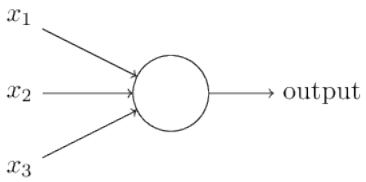

Figure 1: Functioning of a neuron

This is the basic concept of a neural network, but it doesn't really explain how it learns from examples and can solve so many problems.

Figure 1 shows how a single neuron works and Figure 2 shows a neural network where we have an input layer, represented the same way as neurons, but in that layer there is no function being applied, a hidden layer, where we have some neurons, and a function is applied, and an output layer, where a function is applied as well as the final output of the network is given. The lines connecting the neurons represent the weights. A deep neural network is an ANN that has more than one hidden layer.

For the network to learn, an example is fed and an output is obtained afterwards. This generated output is then compared with the desired output (the value that we expect to obtain from the network outcome). We then calculate the difference between the output and the desired result. This difference can be calculated in many ways, using different functions. This is called the cost, or loss function C. It is natural to imagine that we want the cost function with the smallest value, so the output from our ANN is closer to the output that we expected.

With regards to this cost function, its parameters (arguments) are the weights (w) and biases (b), so $C = C(w, b)$, as we cannot change the values of the inputs nor the expected result. Therefore, by changing the weights and biases we can change the output of our network.

Because of that, we can minimize this cost function by setting the most appropriate parameters (weights and biases) of our network. There are many different ways to go about this problem. The minimization of functions is a deeply studied area of mathematics, but the most used technique nowadays for neural networks and deep learning computation is the Gradient Descent Method. This method computes the partial derivatives of our cost function with respect to its variables (parameters), w and b, and uses it to find the values of w and b that minimize our function.

Looking at the network, from the input neurons through the hidden layers of neurons to the output layer we are feeding information forward and analyzing the result, thus the *feedforward* is the first step to understand how an ANN works.

However it seems that we are only looking at the last layer, the output layer, and expecting that by changing its weights and biases our network will learn and be able to give better results. It may happen, but it does not seem to be

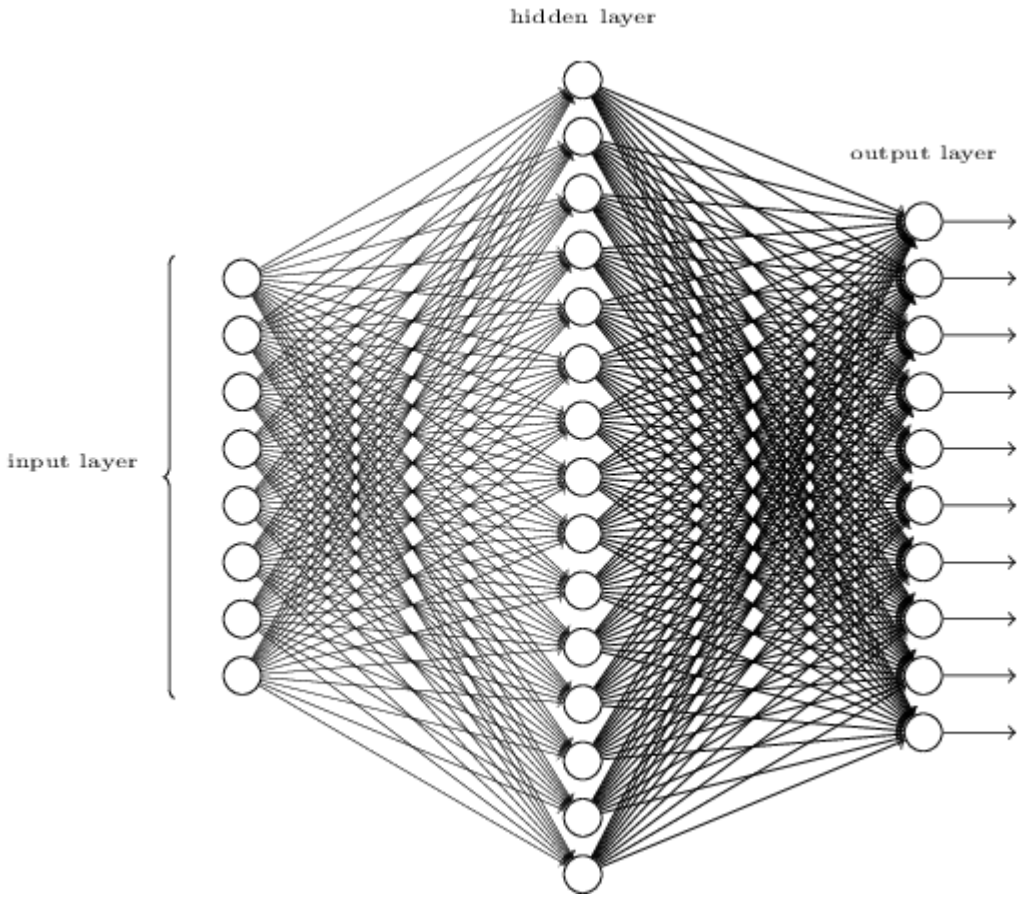

Figure 2: The model of an ANN

a very good method if we have multiple hidden layers, like in a deep neural network. So now we will look at another algorithm that will be more effective to address this problem.

It is intuitive to think that modifications in the weights and biases in the hidden layers of the network will modify the output. Hence, it is desirable to study how changing them affects the output. But also, if it is possible to use the minimization of the cost function to determine how to modify them.

It is easier to see how the weights and biases change the cost function analyzing first the last layer of neurons, the output layer, so we start at this point. Then we go one layer back, do some small changes in the weights and biases and see how it changed the output, then we go another layer back, and so on. That way we are making an analysis from the end of our network to the beginning, we are making a backward analysis.

That is what the *backpropagation* algorithm does, we start by making small changes in the last layer and use it to make another small change in the previous layer and so on. We are analyzing how small changes, δ, in the weights and biases of the network, affects the cost function. So, basically, we are interested in the partial derivatives of our cost function.

$$\frac{\partial C(w,b)}{\partial w} \tag{1}$$

$$\frac{\partial C(w,b)}{\partial b} \tag{2}$$

3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a specialized type of ANN that has great performance for processing data that can be somehow analyzed in a grid-like manner. In at least one of its hidden layers a mathematical operation called convolution is used in place of general matrix multiplication, therefore the name "Convolutional Neural Networks".

Basically, convolution is a mathematical operation involving two functions ($f(x)$ and $g(x)$) that produces a third function ($h(t)$) that express how the shape of one affects the other.

$$(f(x) * g(x))(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx = \int_{-\infty}^{\infty} f(t-x)g(x)dx = h(t) \tag{3}$$

In computer vision terminology the first function is often referred to as the input and the second argument as the kernel. The output is commonly referred to as the feature map.

Usually, when trying to identify or recognize texts, the input to the network is an image. Therefore the input has two dimensions and can be seen as a matrix. That way the convolution happens over more than one axis at a time.

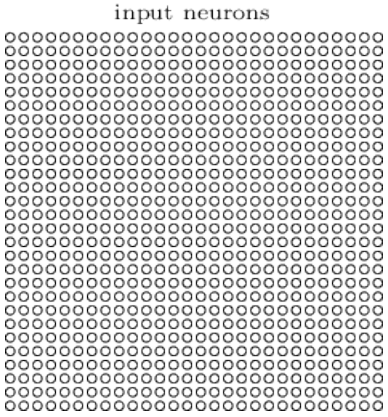

Figure 3: CNN input neurons

That way, if a two-dimesional image I is given as input to a CNN, usually a two-dimensional kernel K will be applied:

$$O(i,j) = (I * K)(i,j) = \iint I(l,c)K(i-l, j-c)dldc \tag{4}$$

As have already been seen in the definition above, convolution is commutative, meaning we can rewrite equation (4):

$$O(i,j) = (I * K)(i,j) = \iint I(i-l, j-c)K(l,c)dldc \tag{5}$$

To help visualize the input of the network think about the images as a $l \times c$ matrix whose entries are the pixel intensity of the image and those values will be in the input neurons, $l \times c$ neurons.

The input neurons have to be connected to a hidden layer. But, unlike in ANN, these connections will not be between every input neuron and every neuron in the hidden layer. Instead, each neuron in the hidden layer will be connected to a small region of the input neurons.

This small region is often called the local receptive field for the hidden neuron (Figure 4). This local receptive field then moves over all the input neurons (Figure 5). For every receptive field there is a hidden neuron in the next layer (Figures 4 and 5).

Notice that this reduces the size of the hidden layers. Another thing that can be done to control the size of the hidden layer, and that also can be used to identify different patterns, is to modify the sizes of the local receptive fields. Such modification should be by how much the local receptive field slides. Its stride length also have this effect over the input neurons.

For each hidden neuron we will use the same weights, bias and activation function. We do that because we want to detect the same pattern, like an edge

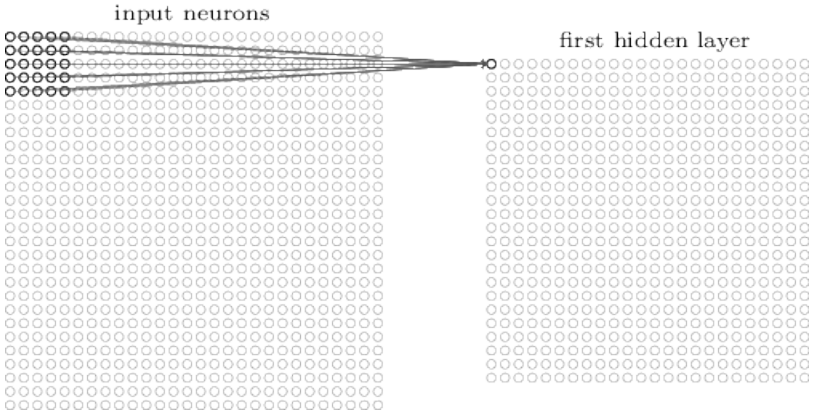

Figure 4: Visualization of a local receptive field

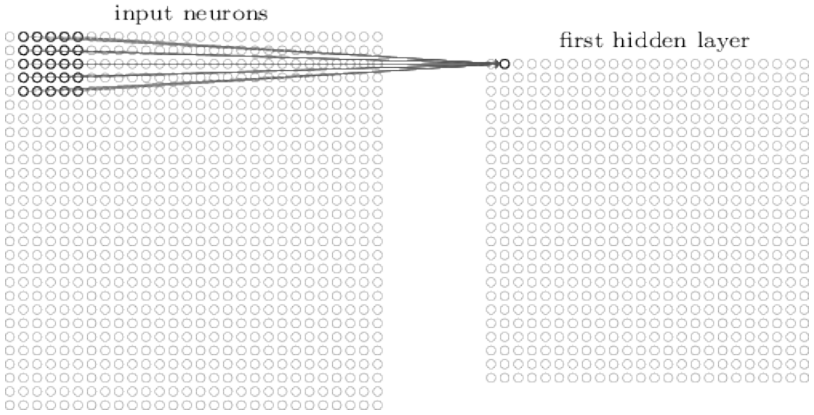

Figure 5: Visualization of a local receptive field sliding over the input neurons

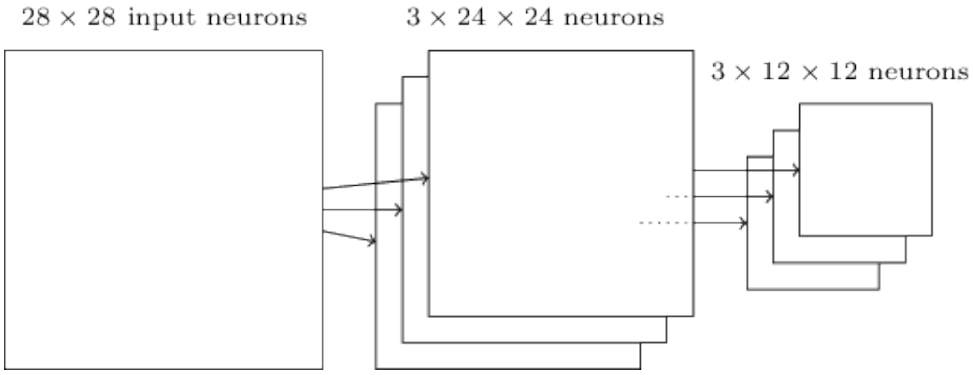

Figure 6: Example of input neurons, convolutional and pooling layers of a CNN

or a horizontal line for example, over all the image. In that way a feature map represents the detection of a characteristic along the entire image.

An image will probably have more than one characteristic that helps to identify it. Because of that, more than one feature map is generated from the input neurons. That is made by changing the weights and biases or even the activation function.

After the convolutional layer, which gives as output the feature maps, there is a pooling layer. In the pooling layer something similar to the convolutional layer happens, i.e, the network looks for features, but this time in the feature map.

That is done with the aim of identifying a way to look for the patterns found by the convolutional operator. This is possible because, once a feature has been found, what matters the most is only its location relative to other features. We can see in Figure 6 that the number of neurons are decreasing.

Finally, we have fully connected neurons. The neurons from the feature maps are fully connected to another hidden layer or directly to the output layer (Figure 7).

That way, CNN are very good for text interpretation as they are very good at identifying patterns. This is true because words are always composed by a finite number of letters, which each one having their own patterns.

4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a type of ANN that can store some of the information that passed trough it to use it afterwards, so there is a loop of information in this type of network. Because of that they resemble more the human brain, as it stores information that was received before to use later. That gives more power and new possible applications to this type of network.

For example, when writing a text it is very important to know if the previous word is masculine or feminine so that we can correctly conjugate the next words.

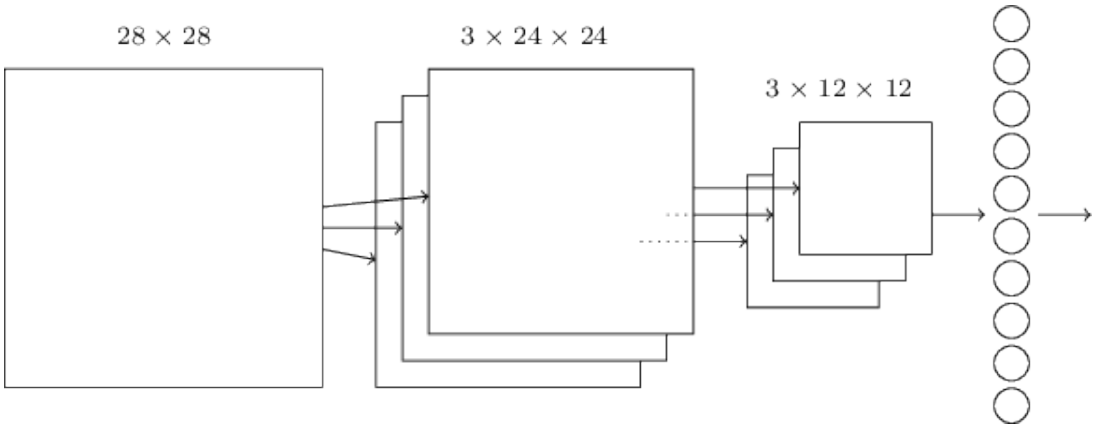

Figure 7: Example of a complete CNN

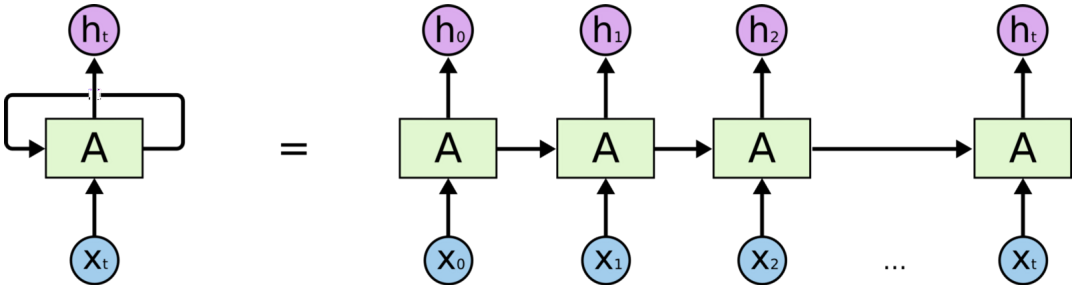

Figure 8: The concept of a RNN

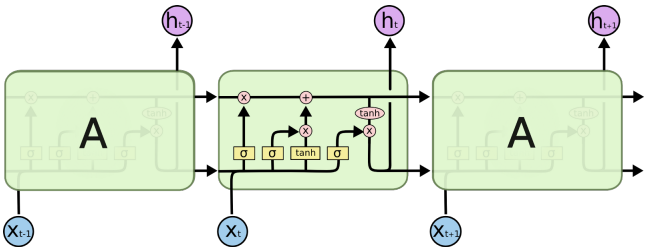

Figure 9: LSTM description

So we can train a RNN to write texts, predict words, describe images and much more.

The process to train RNN is similar to the one used to train ANN: feed-forward, gradient descent and backpropagation. However as RNN keeps information, there is a loop in the network. The gradient calculated and the small changes made to the weights and biases are sent over and over through the network, what can cause a problem in our code.

If the changes made in the weights and biases due to the gradient of our cost function are small, these changes tend to disappear as we keep using the past information stored, because we are sending back and multiplying small numbers over and over. We call this the vanishing gradient problem (remember that these changes that we make are based on the gradient of our cost function). If these changes are large we have the opposite problem, and we tend to get a very large number. We call this the exploding gradient problem.

There are several ways to deal with those problems. We could truncate the backpropagation or add penalties if we reach some specified value, we could find better ways to initialize the variables of our cost function or we can use a Long Short-Term Memory Network (LSTM), which is the most used technique.

Figure 4 shows a diagram of the LSTM algorithm and we can see that the algorithm is not simple. But this algorithm, in a very simplistic description, decides which information that was stored will be used and what new information will be stored for later use. In that way the network can store information for a long period of time without the fear of vanishing or exploding gradient and at the same time it preserves recent information that could be relevant.

Another thing about RNN is that we have some types of it:

- One to many

There is one input to the algorithm and several outputs. For example, an image is given as input and we have a text describing it as output.

- Many to one

Several inputs are given and there is just one output for the network. An example of this is if a text is used as input and as output we have the information

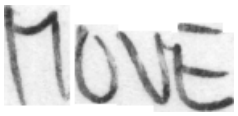

Figure 10: "MOVE" - Example of an image used to train the network

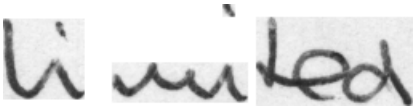

Figure 11: "Limited" - Another image used to train the network

of the emotion behind the text, like if the person that wrote it is happy, writing a positive feedback for a product for example; or mad, writing a negative feedback for a product.

- Many to many

Several inputs are used to generate several outputs. Here we can have a text fed to the network and as output we have the same text but translated to another language. Another example would be a network that learns how to write and as output gives a completely new text based on those ones used to train.

5 Application

From [10] we obtained a very minimalist and simple ANN that contain CNN and RNN. We did not create our own network because the idea of this report was to give an introduction on the concepts behind ANN and some possibility of its applications.

The ANN contains 5 (five) CNN layers and 2 (two) RNN layers using LSTM. The CNN layers come first and extract relevant features from the images. The RNN layers come right after and are in the network as to propagate and keep important information.

This network recognize the text character by character, that way, any word can be recognized by, as long as its individual characters are correctly recognized.

As can be seen in figures 10 and 11 the network is trained using images with cursive and non cursive writing, so that it can identify both in an image afterwards.

6 Conclusion

We used an open source ANN to test how CNN and RNN perform in text detection and recognition. And the results where very good given its simplicity.

	Processing Time	Character Error Rate	Word Accuracy
Training	29 *hours*	13.9%	67.8%
Validation	2 *minutes*	13.7%	68.1%

Table 1: Time that took for our computer (an i3 with 4gb of ram) to train and validate, that is test, the network we used. As well as its performance.

However there are possible ways to improve the accuracy of this network. For example, we could expand the training dataset. Providing new images or even applying filters over the ones that we already have.

Finally, we have that RNN and CNN are two very powerful programming techniques that can be applied to many real world problems. We focused here on their application for text recognition and saw that they can be very good for this problem in particular.

References

[1] http://neuralnetworksanddeeplearning.com

[2] https://www.deeplearningbook.org/contents/convnets.html

[3] https://www.deeplearningbook.org/contents/rnn.html

[4] http://karpathy.github.io/neuralnets/

[5] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[6] http://neuralnetworksanddeeplearning.com/chap2.html

[7] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[8] https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0

[9] http://colah.github.io/posts/2015-08-Backprop/

[10] https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5