Assignment 2 Report – Neural Networks

1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are usually extremely useful in datasets which can be mapped to a spatial distribution where the classification of data relies on features which can be extracted from the interaction of nearby points in this dataset. In most cases, this makes CNNs very appropriate to use with image classification, since they are capable of "seeing" the important features of images (Nielsen, 2017). Nonetheless, they can be used even in cases which don't involve images, but where one is still interested in the properties of neighbouring points in the data. For instance, from the spectrograms of people speaking, it is possible to use a CNN to identify the patterns and phonemes of each language and learn to identify the language being spoken.

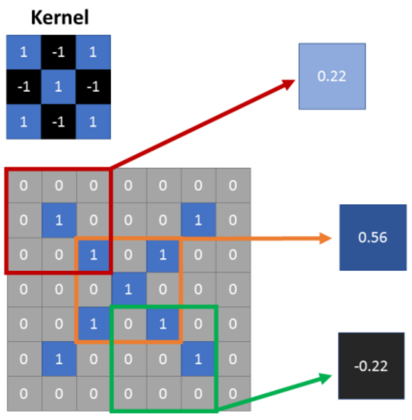

Figure 1 - Demonstration of how a convolution works, in three distinct parts of an example grid shaped like an 'X'. The different values that the convolution takes for this particular kernel in each region of the grid give a sense of how much the feature measured by the kernel is present in that area.

Convolutional layers in a neural network work using a convolution kernel, which is a grid designed to identify features in the image (Figure 1) by superimposing the kernel on each possible part of the image and multiplying its values by that of the image and subsequently averaging them, as well as potentially adding a bias and an activation function (Stanford University, 2015). This procedure effectively generates a new grid (usually smaller, unless a few tricks are used) which contains information about the prominence of the feature the kernel is looking for in different parts of the image. This grid can then be fed into another layer of the network, which can be a usual dense layer of neurons, or possibly yet another convolutional layer, or a pooling layer.

Pooling layers have the objective of reducing the number of points in the resulting grid, and thus reducing the number of parameters to be adjusted during training, as well as avoiding data overfitting. It works by superimposing grids (which can be of any size, but usually 2x2 grids are used) with the image and choosing the maximum value found within that region to represent that area. In this way, the whole area is reduced to the maximum value found within it. This, naturally, greatly reduces the resolution of the image being used, hence resulting in fewer parameters to adjust.

Given this, we have decided to use a CNN on the same problem as treated in the previous report: the classification of 0 – 9 digits in the MNIST dataset (LeCun, Cortes, & Burges, 2013). This way, it is possible to compare the performance of a CNN with the other methods tried previously.

For that, we used the Keras package with a code provided by its own documentation (Keras, 2017), but changing the parameters and layers so as to find a configuration that is both fast and accurate.

The batches used in the network were of 128 vectors and the program ran for only 5 epochs. It used an Adadelta (Adaptive Learning Rate Algorithm) optimiser. The highest accuracy obtained was with the following configuration, in this order:

- Convolutional layer with 6x6 kernels, ReLU activation and 32 output nodes.
- Convolutional layer with 6x6 kernels, ReLU activation and 64 output nodes.
- Pooling layer with a 2x2 pooling grid.
- Dropout of 25% of the nodes.
- 128-node dense layer, with ReLU activation.
- Dropout of 50% of the nodes.
- Dense output layer with Softmax activation.

While this configuration obtained a 99.06% accuracy in the test set after only 5 epochs, it took a long time to run for being computationally heavy. By adding an extra pooling layer between the two convolutional layers and setting the kernel on both convolutional layers to be a 3x3 grid, the computation time was reduced by a factor of 5.5, and with many fewer parameters to tune, it was possible to remove both node dropouts without much risk of data overfitting. With this, the accuracy fell to 98.91% after 5 epochs, which is slightly worse but much faster to run. Hence, we can reach very high accuracies in the MNIST dataset in a few epochs by using a CNN.

2. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are networks which take into account the previous state of the system in order to try and predict the next one. In other words, the output of a layer, along with new information, is fed back into itself, within each timestep. Hence the recurrence. RNNs can thus be used for problems which involve predicting patterns or behaviours, for example, given a sequence of words, the machine might be able to learn from previous examples and complete the sentence in a meaningful way (Britz, 2015). Another problem that can be tackled by RNNs involves, for instance, learning the patterns of profit for a given company through time and trying to predict how it will fare for the next year, given a few assumptions.

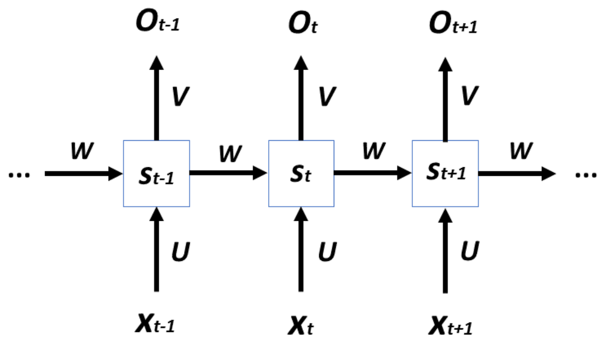

Figure 2 - Scheme of a recurrent layer, at three different timesteps. *W* represents the matrix which takes in the information of the previous state and generates the next one, *U* is the weight matrix that feeds information about the current timestep and adds it to the current state, and *V* the matrix that takes the current state and calculates its output. The s, x and O vectors represent the input, state and output vectors at each timestep, respectively.

A recurrent layer possesses matrices *W, U* and *V* with weights which must be adjusted by backpropagation, much like the other kinds of layers. However, at each new step, the weights are applied to the new data vector $\boldsymbol{x_t}$ that is fed into the system, along with the previous state of the layer, so that these bits of information can interact (Figure 2). At every timestep, an output is thus generated.

A common problem in such a system is that of vanishing or exploding gradients, since in principle there is no limit as to how many timesteps back the backpropagation can go. To avoid these problems, algorithms like Long-Short Term Memory Unit (LSTM) or Gated Recurrent Unit (GRU) are commonly used.

With this, an RNN was applied to the IMBD movie review database (IMDb, 2017) in order to learn how to classify the sentiment of a movie review (as in if it is a "good" or a "bad" review). It analysed only the 5000 most commonly used words and truncated reviews at 500 words, so that they were not too long. The code for this was obtained from the website `machinelearningmastery.com` and subsequently modified by us to test different configurations. The database was comprised of 25000 reviews for the training set and another 25000 for the test set (Brownlee, 2016).

The initial layer structure used by the code was as follows:

- Embedding layer with a dropout of 20% and 32 output nodes.
- Dropout of 20% of the nodes.
- LSTM layer with 100 nodes
- Dropout of 20% of the nodes
- Dense layer with one node with sigmoid activation

Thus, the output was binary ("good" or "bad"), and this configuration achieved an accuracy of 82.98% in the test set over 3 epochs. However, this accuracy could be improved upon by removing the second dropout layer and changing the LSTM layer for a GRU layer with only 50 nodes. This way, the number of parameters to train was greatly reduced, improving the computation speed by about a factor of 2. Furthermore, since the number of nodes in the recurrent layer was smaller, we could do away with the second dropout layer without having to worry too much about overfitting, and thus the accuracy in the test set grew to 87.40%.

The fastest and most accurate configuration we managed to find, however, was as follows:

- Embedding layer with no dropout and 32 output nodes.
- GRU layer with 30 nodes
- Dense layer with one node with sigmoid activation

By greatly reducing the number of adjustable parameters, this structure prevents overfitting and runs much faster (about 4 times faster than the first structure analysed), and managed to achieve an accuracy of 87.68% in the test set.

3. Autoencoders

Autoencoding is when and artificial neural network is used to recreate a given input. It takes a set of unlabelled inputs and after encoding them it tries to extract the most valuable information from them. These are used for feature extraction, learning generative motives of data, dimensionality reduction and compression, since they are capable of taking a potentially highly-dimensional input and, by learning its defining features, encoding it in a form that has fewer dimensions but that has preserved the important information present in the initial object (Goodfellow, Bengio, & Courville, 2016). Autoencoders are based on Restricted Boltzmann Machines and are employed in some of the largest deep learning applications. They are used as the building blocks of deep belief networks. The usual structure of an autoencoder is shown on Figure 3.

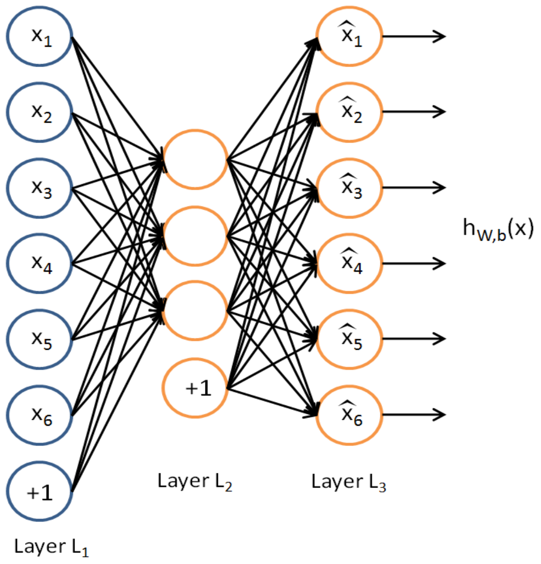

Figure 3 – Simplified schematic of an autoencoder. The input is compressed into a form which has fewer nodes and from this the network attempts to produce an output that is as faithful as possible to the input.

An autoencoder can be divided into two parts. These are the *encoder* and the *decoder*. The encoder needs to compress the representation of an input with a number fo layers. The decoder is then a reflection of the encoder, it works to recreate the input as closely as it can. It has an important role to play during the training of the network, which is to force an autoencoder to select the most important features in the compression. It then uses the method of gradient descent to reach a local minimum of the function. Given this, we have decided to use an autoencoder on the same dataset as treated in the previous report: the images of digits from 0 to 9 present in the MNIST database.

Once again, we used a program present in the Keras package documentation. However, we changed its parameters and layers with the objective of finding a configuration that is fast an still provides autoencoding with a high fidelity (i.e. a small loss).

The batches used in the network were of 256 vectors and the program ran for 50 epochs. It used an Adadelta (Adaptive Learning Rate Algorithm) optimiser and a loss function of binary crossentropy. After the 50 epochs, the autoencoder seems to reach a stable train/test loss value of about 0.11 at a maximum time of 3.01 minutes using the following configuration, in this order:

- Dense encoding layer with 784 nodes, with ReLU activation
- Hidden layer with 32 nodes. This yields a compression factor of 24.5
- Dense decoding layer with 748 nodes, with ReLU activation.

We then added more hidden layers to the network and trained the resulting network again for 50 epochs. The autoencoder seems to reach a stable train/test loss of about 0.10 at a maximum time of 4.94 minutes using the following configuration:

- Input layer with 784 nodes
- Dense encoding layer with 128 nodes, with ReLU activation
- Dense encoding layer with 64 nodes, with ReLU activation
- Hidden layer with 32 nodes.
- Dense decoding layer with 64 nodes, with ReLU activation.
- Dense decoding layer with 128 nodes, with ReLU activation
- Output layer with 784 nodes, with sigmoid activation

It is possible to see that after the input layer, the number of nodes is halved with every new layer. The idea behind progressively diminishing the number of nodes is to find complex interactions between the features so that these can be used to encode the data. However, this comes at the cost of making the program slower, since there are more parameters to train. We see only a slight improvement in the final loss with this architecture, probably due to the process described previously of finding complex interactions between points which can be seen as the defining features of the image, and can only be grasped by a deeper network.

Hence, whether it is justifiable to make the network deeper and thus slower and more prone to overfitting only for a small improvement in the loss will depend on the problem that is being tackled and the fidelity level that is required.

Project: Sunspots and Solar Flares

1. Introduction

Monitoring solar activity is a staple of modern astronomical practice. This is not only because of the scientific prospect of comprehending the underlying stellar structure, additionaly it is done because the Earth is, rather unsurprisingly, greatly affected by the processes that take place in the Sun. Not only do certain warming and cooling periods in terrestrial history seem to be connected to some extent to a variation in solar activity, but also the Sun is constantly releasing highly energetic particles in all directions (Haigh, 2011). Most of these incoming particles are deflected away by the Earth's magnetic field, but in periods of particularly strong solar activity, these events could potentially damage our satellites and telecommunication devices. Hence, it is strategic to monitor solar activity with the objective of trying to predict when such events might occur, in order to attempt to minimise their damage.

Solar flares (Figure 4) are among such highly energetic phenomena. These events are characterized as sudden flashes on the Sun's outermost layers which can be accompanied by an ejection of solar matter into space, called coronal mass ejection (CME) (NASA, 2015). Usually, these strike the Earth about two days after they have left the Sun.

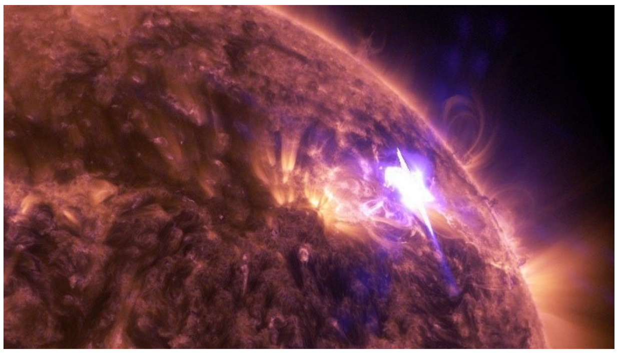

Figure 4 - The occurrence of a solar flare (in bright blue) on the Sun's surface. Picture taken by NASA on the 17th of April 2016.

One of the main drivers of the acceleration of these particles is the turbulent activity of the Sun's magnetic field. As magnetic field lines on the surface connect and disconnect, there are sudden releases of enormous amounts of energy, potentially causing solar flares and CMEs. Another phenomenon that is intimately connected with the Sun's magnetic activity is that of *sunspots*. In places where the magnetic field lines bring convection of the Sun's plasma to a halt, a cold and dark region is formed. These regions, called sunspots (Figure 5), are as much as 3700K cooler than their surroundings and can vary in diametre from a few tens of kilometres to more than 150 thousand kilometres (NOAA).

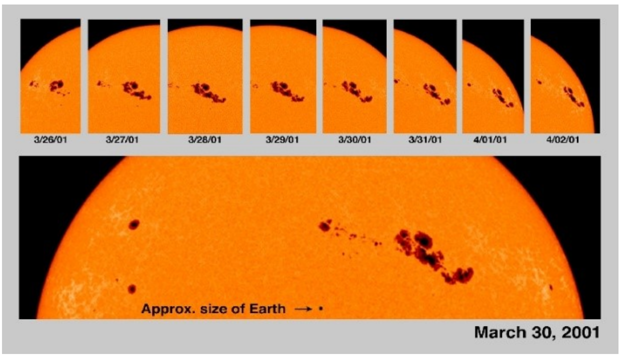

Figure 5 - Images of a large group of sunspots on the Sun's photosphere. Pictures taken by NASA between the 26th of March and the 2nd of April 2001.

The reason why sunspots and solar flares patterns might be treatable by a neural network is that, firstly, they are both related to the underlying magnetic field structure of the Sun. This structure, in turn, largely follows an approximately 11-year cycle during which the Sun's magnetic poles invert. Hence, the objective of this project was to 'teach' a recursive neural network to understand this magnetic cycle through sunspot data and predict future sunspot values and potential solar flares.

2. Part I – Sunspot Cycles

It might not be at first a straightforward task to define 'sunspot activity' on the Sun's surface. Hence, astronomers have defined the *sunspot number* at a given time as

$$R = N_s + 10\, N_g \qquad (1)$$

where R is the sunspot number, N_s the number of sunspots present on the solar disk and N_g the number of sunspot groups on the solar disk.

With this definition, it is possible to find average monthly sunspot number values dating back to the 18th century. We have obtained the data in the website of the Solar Influences Data Analysis Center (SIDC) (SILSO, 2017). The data used for the first part of this project can be visualised on Figure 6.

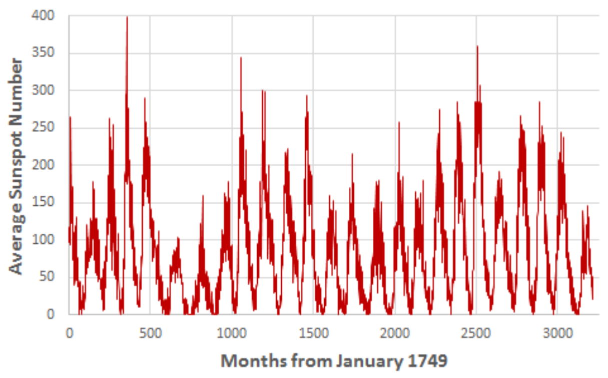

Figure 6 - Average sunspot number measured from historical records. Months are counted from January 1749 onwards. The last data available is September 2016.

From this figure, one can easily see the 11-year cycles the sunspot number undergoes. However, to make this more precise, and to compare our predictions to expected values, we have performed a discrete Fourier Transform on this data, which can be seen on Figure 7.

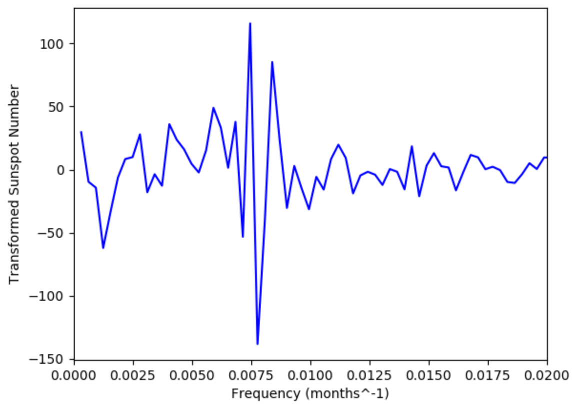

Figure 7 - Discrete Fourier-transformed sunspot data. One can clearly see a dominant frequency peak, which represents the 11-year magnetic cycle of the Sun.

From this, we have obtained that the dominant frequency (i.e. where the main peak of the Fourier-transformed data is) was of 0.00746 months^{-1}, which amounts to a period of about 134.05 months, or approximately 11.17 years. Furthermore, from the original data itself, we have obtained that the standard deviation of the period between peaks is of 27 months, or 2.25 years.

With the objective of predicting sunspot number patterns, we then proceeded to create an RNN which took in an array with n elements which represented the sunspot values of the n previous months. The network then attempted to predict what the sunspot value for the following month would be. The layer structure used was rather simple:

- LSTM layer with 20 nodes
- Dense layer with 1 node

Deeper architectures than this one have been tried. However, they did not perform better and were much slower. Presumably, this could have been caused by data overfitting, but also because the pattern of sunspots on the solar surface might not be heavily dependent on complex interactions between values at greatly different time frames. Rather, it is more likely a stochastic evolution that depends mostly on the values it took a few steps (in this case months) before. Hence, the improved potential precision of adding more layers does not compensate the exponentially growing number of training parameters and the accuracy is thus not increased.

The optimiser used was *Nadam*. This optimiser is the usual RMSProp (which is based on gradient descent and appropriate for RNNs) with added Nesterov momentum for better convergence. This optimiser also uses the Root-Mean-Square Error (RMSE) as the value to minimize in the gradient descent. It is defined as follows (Holmes, 2000):

$$RMSE = \sqrt{\frac{\sum_{i=1}^{k}(\hat{y}_i - y_i)^2}{k}} \qquad (2)$$

where k is the total number of measured values (or instances) to sum over, $\hat{y}_i$ the value predicted by the machine for the ith instance, and y_i the correct value for that instance.

The data was split into a train and test set, where the training part was 70% of the initial data and the test set the remaining 30%. The results for this system are as follows: by looking back 50 months (i.e. n = 50), with mini-batches of 10 vectors and after 30 epochs the network reached a 1.44 RMSE at predicting the sunspot number for the next month in the test set. In the following graph (Figure 8), one can compare the machine's predictions with the original data.

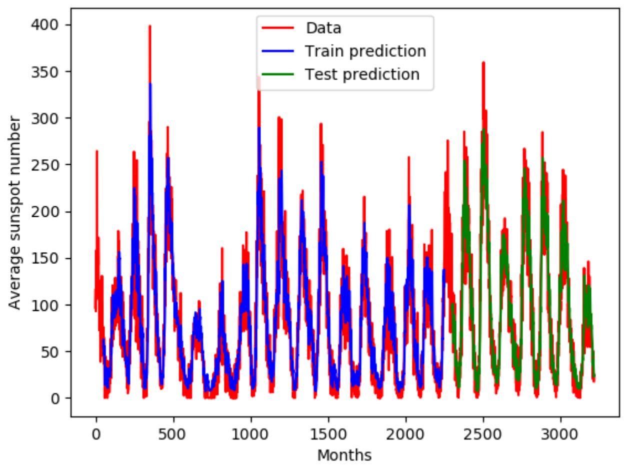

Figure 8 - The network's output for the train and test sets when fed each of the values for the last 50 months in the data and asked to predict the following month.

Both numerically and visually, one can see that the network is very successful at predicting the sunspot number one month in advance when given data for the last 50 months, since from the graph the predictions and data have a large overlap and the RMSE value obtained is rather small when compared to typical values the sunspot number reaches. In fact, one can calculate the Normalised Root-Mean-Square Error (NRMSE) of this result to be of about 0.36%, which is rather small, where the NRMSE is simply the RMSE divided by the difference between the maximum value of the sample and the minimum value of the sample, and it is often expressed as a percentage.

Then, one can try to feed the network only the predicted values for the previous months instead of using the real data as well, and when this is done the network quickly starts to give off inconsistent and rather bad results. It is believed that this happens because the errors in each prediction accumulate and interact chaotically after every iteration. To solve this to some extent, we have then changed the network so that, when fed the values of the n previous months, it would give predictions for the next m months, all at once. In that way, the expected error for one specific month might be increased, but predicting months in batches avoids the accumulation of errors for every subsequent month by treating every month in one batch on an equal footing, and generates a prediction that is much more consistent with reality for the whole batch. The architecture for this is only slightly different:

- LSTM layer with 20 nodes
- Dense layer with m nodes

It was also necessary to change the objective output vectors in the train and test sets for this purpose. When we use n = 50 and m = 50 and training the network for 50 epochs, the NRMSE value returned is of 1.43% in the test set. While, as expected, the error has increased when compared to the network that only predicted one month, it is still a small value, and in this case the network can predict 50 months in advance without a fast accumulation of errors as one looks more and more into the future.

It is very tempting, then, having a network which seems to predict the sunspot numbers so well in advance, to feed it values for the last months of available data and ask it to predict the future months. Thus, this is what we then did. We fixed m = 150 and trained the network for different values of n, to see how far back it is best to look at in order to predict future sunspot patterns. The RMSE values for the test set are shown in Table 1 for such different values of n, along with its values for the next sunspot peak. Furthermore, Figure 9 depicts the evolution of sunspot numbers predicted by the network for the different values of n (steps back).

Table 1 – Results of the network trained to predict the 150 months following September 2016 (last available data). The RMSE is measured from past, available data.

n	**NRMSE (test)**	**Next peak month**	**Next peak value**
10	2.53%	59	110.4
50	1.93%	75	129.1
100	2.36%	100	70.8
200	2.36%	107	88.2
500	1.57%	90	99.6

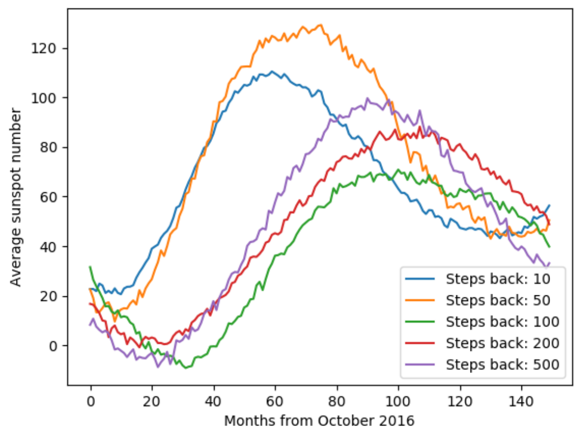

Figure 9 - Sunspot number predictions of the network for the 150 months following September 2016, using different values for the number of past months looked at for making the prediction.

Naturally, since this is an attempt at predicting the future and hence there is no data available to compare it to yet, it is not possible to tell how accurate these predictions are. However, since the network which used $n = 500$ had the smallest NRMSE value when compared to past data, let us assume that it is the most accurate configuration. In that case, it predicts that the next sunspot number peak value will occur 90 months after October 2016, which is April 2024. If we suppose that the current solar magnetic cycle will evolve as usual (and there is no evidence to believe otherwise), we can take the last peak available in the data, which is February 2014, and sum to it the expected peak interval we calculated from the Fourier-transformed data, which is of 134 months, and we obtain that the next expected peak should be in November 2025. There is a difference of 19 months between these two calculated points. However, these values lie within one standard deviation (27 months) of each other, and thus we can say that they are statistically compatible, which is good evidence that the sunspot number evolution predicted by our network is accurate to a large extent.

3. Part II – Solar Flares

Next, we obtained the data concerning the solar flare events from the Fermi GBM (Gamma-Ray Burst Monitor) catalog, hosted by NASA (NASA, 2017). The data provide the days in which solar flares happened and the intensity of each flare, measured in photon counts. The data begins on the 2nd of November 2008. We then shaped the data to fit our needs by counting how many solar flares were detected on each day since and what the average solar flare intensity was for each day. Since now we are dealing with daily variations in data (as opposed to monthly variations in the previous part), the graph for the whole period looks particularly noisy, and it would not be very instructive to show it here. However, in the next Figures we show together the normalised values for daily solar flares, average intensity and sunspot number. The daily values for sunspot number were obtained from the same source as for the monthly values analysed in the previous section. On Figure 10, we show these normalised values for a 100-day interval in the data. On Figure 11, we show the whole interval of data, but for clarity we have taken a 30-day moving average of the values so that the patterns can be observed more clearly.

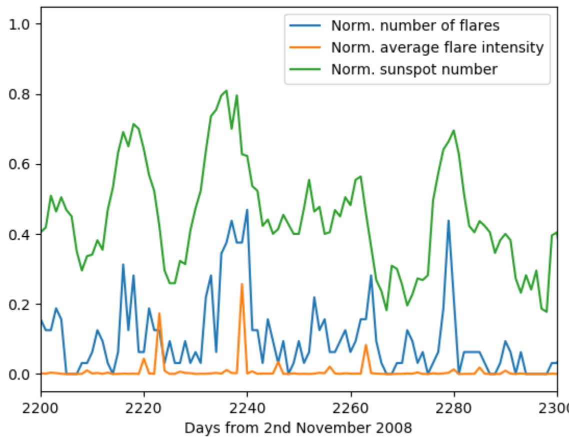

Figure 10 – Sunspot number, flare intensity and number of flares in a 100-day interval of the data. Each of the sequences has been normalised to lie between 0 and 1, since there are very considerable differences in the orders of magnitude of these variables.

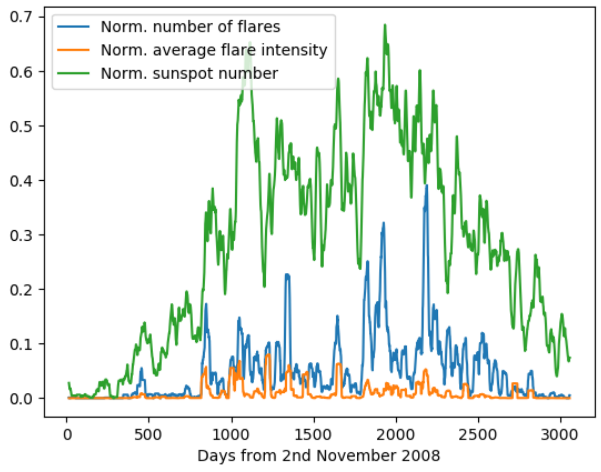

Figure 11 - Normalised values for the entire period, with a 30-day moving average taken so that the patterns can be compared.

It is possible to see that there has been an increase in solar activity in the middle of this interval, which translates in more numerous flares and higher average sunspot numbers.

This time, we fed the network the sunspot numbers, flare numbers and average flare intensities from the previous *n* days and trained the network so that it would predict the solar flare numbers and intensities for the next *m* days. This time, we did not require it to predict sunspot patterns since this has already been done in the last part (albeit for monthly values), and would increase the number of training parameters, making the program potentially less accurate and slower. Much like in the last part, we started by requiring that it only predict the following day (i.e. $m = 1$). The architecture used for this is as follows:

- LSTM layer with 20 nodes and a dropout factor of 20%
- GRU layer with 6 nodes
- Dense layer with 2*m* nodes

Heuristically, since now there are more coefficients to adjust and predict, there is a higher probability of overfitting the data, hence why there is a dropout of 20% in the first layer. Furthermore, the idea behind having two recurrent layers this time is so that the first one can identify the most important attributes in the interactions of values for distinct days and boil it down to simpler rules, hence why there are fewer nodes in the second layer. This was not necessary in the previous part because there was only one kind of input variable, whereas this time there are three. The optimiser used in this network was the same as before: *Nadam*. Also, once again 70% of the data was used for training and the remainder for testing.

When one runs this network for 30 epochs with mini-batches of 20 vectors and with $n = 50$, the results obtained are as follows for the $m = 1$ case (Table 2):

Table 2 – Results of the network's predictions for the two quantities analysed.

	Train NRMSE	**Test NRMSE**
Number of flares	11.05%	18.04%
Average flare intensity	8.3%	6.61%

Figure 12 shows the machine's predictions compared against the data for the

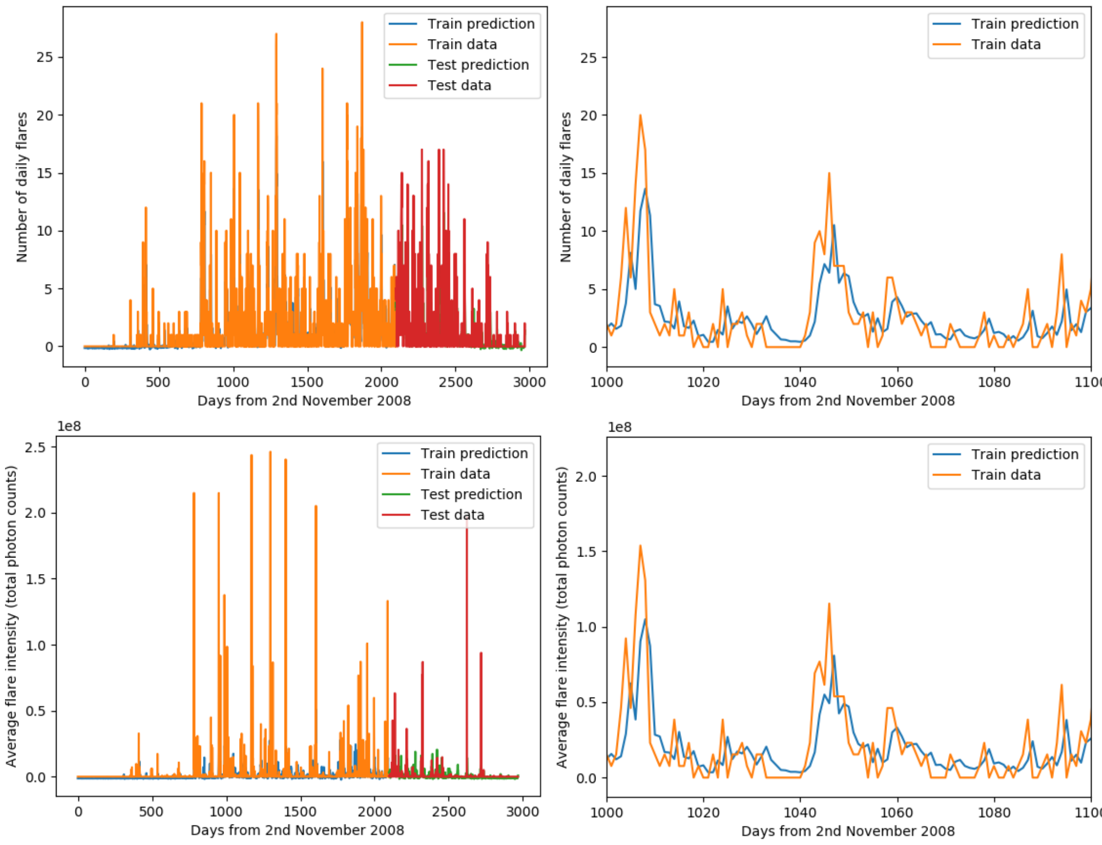

Figure 12 - Comparison between the network's predictions and the data for the number of solar flares and their average daily intensities, exhibited for the whole data interval and for a 100-day span.

whole period and for a specific 100-day interval, for clarity.

From the table, it can be noticed that the NRMSE values are considerably higher for the prediction of solar flare numbers and intensities than for the sunspot values in the previous section. Firstly, it must be highlighted that the sunspot values of the previous section were monthly averages, and thus the data becomes much more smooth and predictable. This can be seen when we look more closely to the 100-day interval and compare data with predictions. In both flare number and intensities, what usually happens is that there is a peak which the network fails to predict. Then, on the following day the prediction catches on with the trend, usually with lower values of flare number and intensities. The final effect of this is an impression that the predictions are always delayed by one day. This probably happens because the network realizes that when there is a solar activity peak on a particular day, then the following days are likely to also yield intense solar activity. However, the network cannot seem to be able to predict the peak that triggers this process in the first place, and thus the best it can do is try to catch on once it happens.

Naturally, considering that the errors were already quite high when the network tried to predict one day in advance, it is expected that the NRMSE values will only grow when we increase *m* and get the machine to try and predict a batch of days in advance. For completeness, then, Figure 13 shows this error evolution when we varied the value of *m*, with *n* = 50 and where each run involved 30 epochs with mini-batches of 10 vectors.

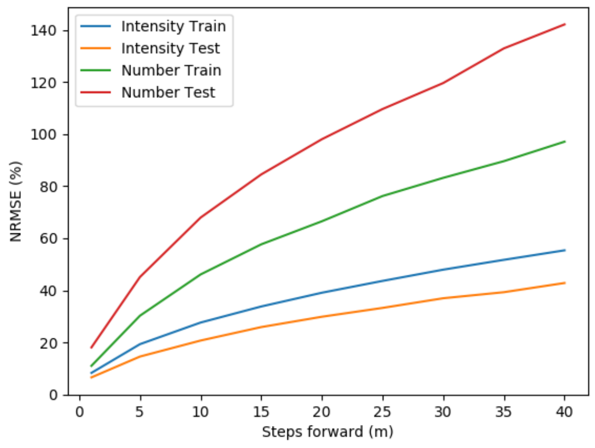

Figure 13 – The evolution of the prediction error for the two variables (flare intensity and flare number) in the train and test sets, when one varies the number of days in advance the network tries to predict.

While it should come as no surprise that for all cases the error increases monotonically when one tries to predict more and more days in advance, it is rather curious that the set with the lowest error (yellow line) is a test set. One would normally expect that the errors would be higher when one is testing the data, which is what happened with the flare number variable. We believe that in the specific case of the flare intensity, this happened because the interval which was used as the test set is probably "better behaved" than the train set. That is to say, there is a smaller density of very high peaks which the network fails to predict and which considerably increase the error. It can be seen, for example, that this is not the case for the flare number variable. Both the train and test sets contain roughly the same density of sharp changes and high peaks, which explains why this variable yielded the expected relation: the test set has a higher error than the train set.

Thus, it seems that the daily flare data is too noisy for the network used to give accurate predictions, and furthermore it can only give predictions with a relatively low error rate one day in advance. This leads us to believe that while the general trends of sunspots and solar flares are somewhat predictable and cyclic, their specific behaviour day after day is too chaotic and unpredictable, probably because it depends on extremely complex stochastic interactions happening in the outer layers of the Sun.

As a final attempt, then, we can observe what happens if one tries to smooth out the data to capture mostly the general solar flare trends over time. For this, we used the same network, but this time it was fed values for which a 30-day moving average had been taken, much like the one shown on Figure 11. So as to create a valid comparison, we used again *n* = 50. When asked to predict the following day (*m* = 1), the network obtained the results on Table 3.

Table 3 – Results for the smoothed data.

	Train NRMSE	Test NRMSE
Number of flares	6.37%	5.84%
Average flare intensity	8.24%	7.88%

Indeed as expected, the errors are mostly lower when the flare and sunspot data has been smoothed out to highlight the general trends. This serves as further evidence that the high errors being found in the previous attempts at predicting the solar flare patterns are due to the erratic nature of these events.

4. Conclusion

In this project, we attempted to create a neural network which can make predictions pertinent to the events that take place on the surface of the Sun, namely sunspots and solar flares, which are related to the activity of magnetic fields which interact with the plasma present there. The first part of the project aimed at predicting monthly trends in sunspot activity by training the network with data that dates back to the 18th Century. As was shown, the network is indeed very successful at grasping the 11-year magnetic cycles of the Sun and has a low error rate even when trying to make prediction of entire batches of months at once. The predictions made not only match with the available data to a large extent but are consistent with what we theoretically expect from the as-of-yet unavailable data (future months).

In the second part of the project, we aimed at trying to predict solar flare patterns, namely how many are expected to take place on a given day and how intense they will be, on average. Even when the network was fed information from the previous 50 days and asked to predict the following day, the network struggled to make accurate predictions, and as was discussed, this is probably due to the somewhat chaotic behaviour of these events taking place on the Sun's surface. Even though there are general trends which can be observed over the years (as the first part of the project showed), making accurate predictions for specific days is complicated due to the high number fo factors involved and the complex interactions that give rise to these events, much in the same way that one can study geological trends in climate over thousands of years, but it might still be rather hard to predict the weather at a specific place one week in advance.

Thus, we conclude that the Recurrent Neural Networks used for this project are very handy and accurate when predicting long-term trends in solar activity, and can be used to model these trends and foresee future behaviour. However, even though they can somewhat predict daily patterns one day in advance, in order to obtain more reliable and accurate results we believe that the model needs to include more variables so as to be able to take into account a more complex instantaneous behaviour.

Bibliography

Britz, D. (2015, September). *RECURRENT NEURAL NETWORKS TUTORIAL, PART 1 – INTRODUCTION TO RNNS*. Retrieved from WILDML: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Brownlee, J. (2016, July 26). *Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras*. Retrieved from Machinelearningmastery.com: http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Chapter 14 - Autoencoders*. Retrieved from Deep Learning Book: http://www.deeplearningbook.org/contents/autoencoders.html

Haigh, J. (2011, February). *Solar influences on Climate.* Retrieved from Imperial College London: https://www.imperial.ac.uk/media/imperial-college/grantham-institute/public/publications/briefing-papers/Solar-Influences-on-Climate---Grantham-BP-5.pdf

Holmes, S. (2000, November 28). *RMS Error.* Retrieved from Stanford Department of Statistics: http://statweb.stanford.edu/~susan/courses/s60/split/node60.html

IMDb. (2017). *Homepage*. Retrieved from IMDb: http://www.imdb.com/

Keras. (2017, March 26). *Keras Examples*. Retrieved from GitHub: https://github.com/fchollet/keras/tree/master/examples

LeCun, Y., Cortes, C., & Burges, C. J. (2013, May 14). *The MNIST Database of Handwritten Digits*. Retrieved from Yann LeCun: http://yann.lecun.com/exdb/mnist/

NASA. (2015, July 31). *Sunspots and Solar Flares.* Retrieved from NASA: https://www.nasa.gov/multimedia/imagegallery/image_feature_2201.html

NASA. (2017, April). *Fermi GBM Flare List*. Retrieved from Reuven Ramaty High Energy Solar Spectroscopic Imager: https://hesperia.gsfc.nasa.gov/fermi/gbm/qlook/fermi_gbm_flare_list.txt

Nielsen, M. (2017, January). *Chapter 6*. Retrieved from Neural Networks and Deep Learning: http://neuralnetworksanddeeplearning.com/chap6.html

NOAA. (s.d.). *SUNSPOTS/SOLAR CYCLE.* Fonte: NOAA: http://www.swpc.noaa.gov/phenomena/sunspotssolar-cycle

SILSO. (2017, April). *Sunspot Number*. Retrieved from Sunspot Index and Long-term Solar Observations: http://www.sidc.be/silso/datafiles

Stanford University. (2015, January 7). *Convolutional Neural Network*. Retrieved from UFLDL Tutorial: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/