Musical Instrument Recognition using Neural Networks

Computational Intelligence

Submitted by

Names of Students	E-mail (@est.fib.upc.edu)
Andrés Flores Valle	andres.flores
Miguel Méndez Pérez	miguel.mendez.perez

Universidad Politecnica de Catalunya
Master in Artificial Intelligence

Abstract

In this paper a system for musical instruments recognition is presented. A retrieval of the main components of different audios has been done, in order to teach a Neural Network to distinct between the noise and the characteristics of each instrument. The performance of this work has been validated using around 10000 audio samples of a total of ten different instruments, played by different people and using different techniques. The results are analysed and compared with other works, some possible future improvements are given.

Keywords: Neural Networks, Features, Feature Extraction, Music Recognition, Fourier Transform, Acoustic Signal Processing, Frequency, Pattern Classification.

Contents

1 | Introduction

The goal of this project is to recognize musical instruments, given and audio file recorded with an isolate note, by using Computational Intelligence techniques, more concretely Neural Networks.

Speech recognition has been received much more interest along the last years than Musical Instrument Recognition, however the last one would benefit music software such as record software programs or even benefit amateurs and professional musician.

Recognition of musical instruments is a hard task that has been attempted to solve in the last thirty years. Musical instruments vibrate, creating sound waves that reach our ears, this vibration is converted to nerve signals which allow to our brain to recognise different sounds. How does it accomplish this task? The brain recognises various characteristics of sound, i.e. some features that can be associated to different objects emitting sounds, such as the frequency, the amplitude or the waveform.

For this work we will focus in the previous idea. The waveform is different for each instrument, even if they are playing the same note, i.e. the same frequency. It is known that each musical instrument produces a different wave sahpe depending on its characteristics, e.g. in the case of a guitar, features such the wood, the strings, the layout, etc. For this reason, it is not a simple task to recognise a concrete instrument, because even though two instruments are of the same type, both have different features. Therefore the waves produced by them are very similar but not identical. We will use this fact in order to apply a neural network which allows us to solve this ambiguity.

Automatic music instrument recognition and classification has be done in many ways and applied in music information indexing and retrieval. In general, the methods implemented nowadays have to deal with these three important aspect: decision of the signal of the data, choice of its representation and extraction of features, and the classification algorithm. The signal of the data can be classified in three different classes: Signal consisting in isolated instruments playing a single note (SISN, the simplest ones); signal of

isolated instruments playing multiple notes (SIMN, which are the more complex); and signal of multiple instruments playing multiple note (MIMN, the most complex) [14]. The usual signal choice of the data is the simplest one, SISN, the same as used in this project. The feature retrieval or representation is done in simple ways considering only the frequency of the signal [10], or complex ways considering spectrum features [9], [4]. The features chosen in this project are given by the harmonics of the instruments wave. We describe this procedure in the appendix A. Classification algorithms such as neural networks are a reasonable choice since the data is usually noisy. Neural Networks have provided good results for classification of instruments, [3], although the $K - NN$ algotithm, HMM, $Kohonen$ or SOM methods are also used [10]. In this project we will consider neural networks in order to analyse their behaviour in the classification.

2 | The CI methods

2.1 Neural Network Layout

The structure of our network has changed along the development of our system, different approaches has been applied in order to increase its performance.

At a very early step we decide to use a network composed by three layers, where the first one of them corresponds with the data input. The second layer is called the hidden layer and the number of neurons selected for it has been deeply study, many different approaches have been found, which use to be based on rules of thumb. A very common one relies on that the optimal size of the hidden layer is usually between the size of the input and the size of the output layers. This rule is defended in many books and articles as Introduction to Neural Networks with Java [6]. Some experts criticize them because they are not taking into account important factors as the number of training instances or the amount of noise in them. Another rule has been found which tries to solve this problem, in this cases the relation between the training cases and the number of weights wins importance. As we know, if the number of weights of our networks is approximately as big as our training examples, fitting will be an easy task. As John Von Neumann said: "*with four parameters I can fit an elephant, and with five I can make him wiggle his trunk*".

Hence, the problem of fitting will be easily solved but it will lead to an over-fitted result, so our system is just adapting to the training instead of learning. A typical recommendation is that the number of weights should be around 1/30 times the number of training instances. It seems at a first glance that those rules following this second approach are better that the previously explained ones, but when we add regularization to our network, the problem of overfitting is highly reduced, so we do not have to be as worried with the number of weights as before. Lawrence et al. [8] have shown that, using online backpropagation, increasing the number of weights can help to find a good global optimum. After more reading, we arrive to the conclusion that

there are multiple theories and each one defends its own one, but it is clear that there is not a perfect way to estimate the perfect number of hidden neurons yet, so testing different configurations is needed.

For the output layer the number of neurons was subjected to the number of instruments we want to distinct. For instance, if the training corresponds with piano and violin audio samples, the output layer will have two neurons. The activation of each one will depend on the input, this is that one neuron will be activated when the input is a piano sample and the other when a violin audio is entering to our system. Hence, the number of neurons in the final layer will be equal to the number of instrument we want to recognise.

2.2 Methods

The methods that has been applied to our network were selected taking into account that what we were building is a classifier system, since our goal is assign to each input or audio, its correspondent output or instrument.
In order tho achieve this, a pre-processing of the data must be done first, such as find a suitable representation of the data for the network input. This is shown in detail in the appendix A. Then we implement a basic neural network and we add new components to it. The full development can be found in B, where we explain the different steps we have followed: why decide about incorporate all the parameters and functions to our network and where we found the information to carry this out.

3 | Results and Discussion

In this chapter we will explain how our system works and the different test that we have carried on. In order to find the best accuracy, different experiments has been done, over different configurations, trying to find the best combination of hyper parameters. The task of testing the different combinations is tedious and complex, because the possible combinations are very large. To ensure that our results are statistically reliable, we execute each configuration 50 times, increase this number would suppose in a huge amount of cputime. We have to think that, for each iteration the 5-fold cross-validation algorithm is applied, which implies to train the network a total of 5 times per iteration. This results in a total of 250 trainings, which can be done with different epoch number. The computations behind this are incredibly vast.
The network will be trained a total of 5 tunes, but in each one of them we will reserve one of the subsets, this is that will not collaborate in the training and it will be used to validate the training. In this way, we can obtain a metric about the training process.

We have divided the experimental part in three sections. In the first we will use different recordings from a total of 9 instruments playing the same note. The justification of this is because the harmonics used in order to represent the data vary for different notes in the same instrument. Therefore the first approximation of our approach is to classify the instruments playing the same isolate note, since the harmonics will be similar for the same instrument. The problem of using this approach is that the number of samples is very reduced, about 10 samples for each instrument. The second test is more complex, consisting on samples of two instruments but playing the full set of notes that each specific one can play. This is the most logic step forward in our analysis, since now the harmonics can vary a lot for the same instrument. In the last test we increase the number of instrument until a total of eleven, trying to see if our approach for the representation of the data is enough for discrimining the classification. Each instrument will play its total spectrum of notes. The last test is the most complex one, since the harmonics may not

be suitable in order to get a good performance.

3.1 Result for 9 instruments set, same musical note

In this experiment the different instruments will play the note A3. This note has been chosen because it is commonly know in tuning and we have enough samples of it. This was the starting point of our project, multiple experiments has been carried out here in order to find the best results. The total of samples is just 121, so more audios would be needed in order to ensure the consistency of the network, although its free acquisition is very hard.

A deep analysis has been performed over this data, where multiple different combination of hyper parameters were applied. The resulting cost and image resulted graphics for each combination are stored in Figures directory. We have resume in a table (see Table **??** in the appendix) the accuracy results, in green we have highlighted those cases where the accuracy is maximum and in orange the cases where a clear overfitting was detected or the learning was not feasible. An example of this is presented in Figure 3.1, where the overfitting can be seen clearly. The training cost raises where the validation goes down with the number of epochs.

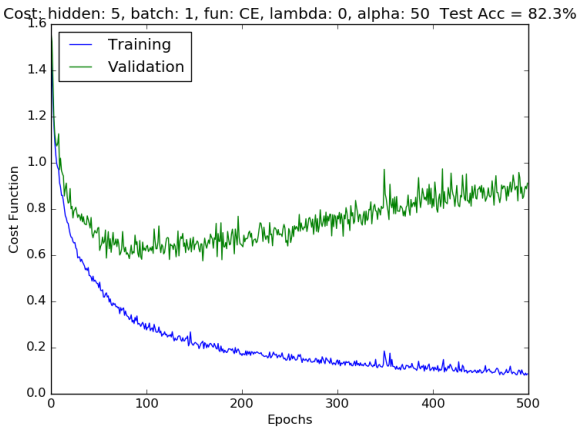

Figure 3.1: Overfitting example

3.2 Results for 2 instrument set, different musical notes

In this section we will describe our results over a dataset composed by two different instruments, flute and viola. The reasons for using this instruments are that we dispose of almost the same number of samples of each one, 548 for the viola and 582 for the flute. The resumed information about this dataset can be found in the code, in "data/sets" directory. Anyone can distinct between a flute and a viola without problems and this fact is applied also in the physic of the harmonics, so sonorous waves of both instruments are different, which should make the learning feasible.

Let's compare some of the results we have obtain. The first hyperparameter to check is α, also called learning rate, which is one of the most important ones in our network. This value is the one which will set how much are we changing the weights and bias of the network. Hence, a good selection of it, is fundamental if we want to obtain good results from the net. In Figure 3.2 we can see both the accuracy for the 2 instrument set. The accuracy for the testing is 53.5%00, the test data is that one which was maintained away during the training, so it is not contaminated. The learning with this parameters is null, although the cost function value is decreased. Maybe with more epochs we could have found a better solution, but still, this value of α is probably too small. For this we will increase this parameter and observe the new results, which are collected in the same figure in the center graphic. As we clearly see, in this case the network is learning and the accuracy is increased 65.6%. What is happening with the α parameter corresponds with what we said before. Nevertheless, in the right graphic, we reduce the regularization term to zero and as a result of this we achieve better accuracy. The regularization term aim is reduce overfitting, but in this case is damaging our results, because the data is well-formed and with only 200 epochs our system is not overfitting, this a good example of the bad application of regularization.

When α is too large, we probably will skip the optimal solution, during each epoch the applied changes to the weights and biases will make the cost jump as it can be observed in the Appendix in Figure C.1.

Now we will analyse how the hidden layer neurons affect to the accuracy of the system. The number of hidden neurons will give more complexity to our system, one way to check if more neurons are necessary or not is comparing the accuracy results and observing if both, accuracy and neurons

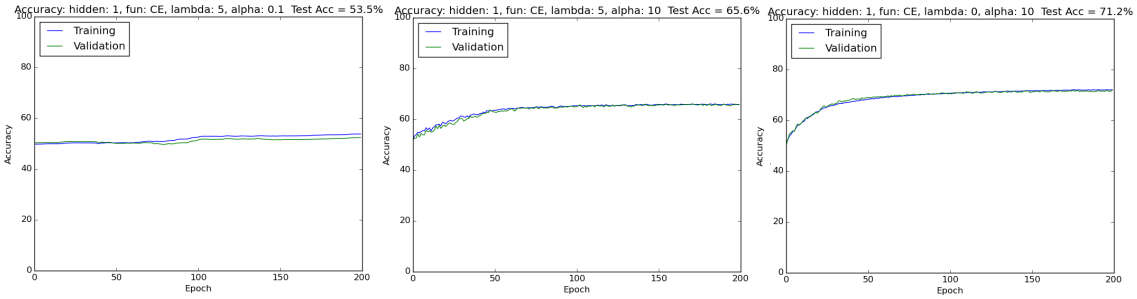

Figure 3.2: Accuracy for different α

are directly proportional. A summary of this experiment can be found in Figure 3.3, where we show three different plots. The hidden neurons from left to right are 1, 8 and 20. The three plots are very similar, we see that for the first case the accuracy raises during the first epochs and then show horizontal asymptotic behaviour. For the second case, the system learns faster during the first epochs and then shows the same behaviour. In the last case, where the hidden neurons are 20, the accuracy increases a lot in the first epochs and we can observe that during the last ones is still increasing, so the network is still learning. The cost results are collected in Figure C.2 in the Appendix, where it can be observed how the different systems learn without fall in overfitting.

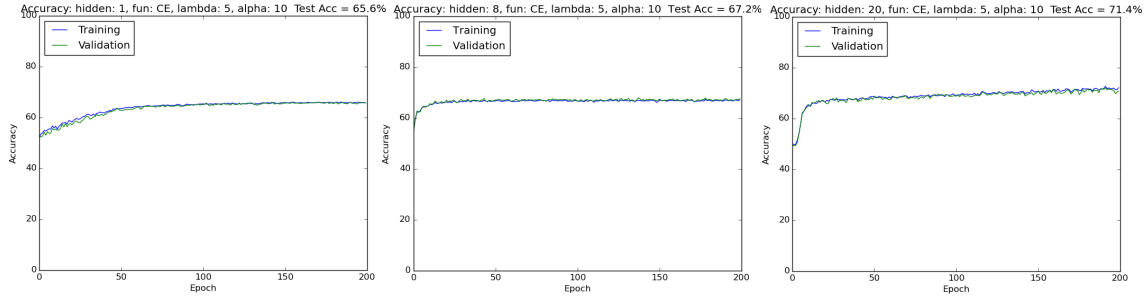

Figure 3.3: Accuracy for different number of hidden neurons

Therefore, the number of hidden layers can improve the performance of our system, but after experimental results, having more than 20, does not reflect too much in the accuracy, so we will conserve this value as a reference for the system.

As it was said before the regularization parameter, seem to behave in a strange way for this cases. Let's analyse some results in order to obtain

clues about its behaviour. In Figure 3.4, three new graphics are plotted, the hidden neurons is 20 in all of them and α parameter is 10. Lambda value will be different on each one, for the left graphic will be 0, for the middle one 1 and for the right one is 8.
Surprisingly, or not, the best accuracy is found in the first graphic, a 73% which is the highest value we have reach. How is that possible if regularization term is zero? We have said that regularization was helping us to reduce the overfitting, well this is true. The problem is that our network is not overfitting, what we are doing adding the regularization term is adding more restriction to the cost functions and to the learning in general. So we affecting the accuracy. We have applied L2-norm regularization, which tries to keep a slow value of weights in order to find the simplest hypothesis over all our set. Then we can assume that our system needs to be complex in order to solve this classification problem and regularization is harming it. (Cost functions can be found also in the appendix in Figure C.3)

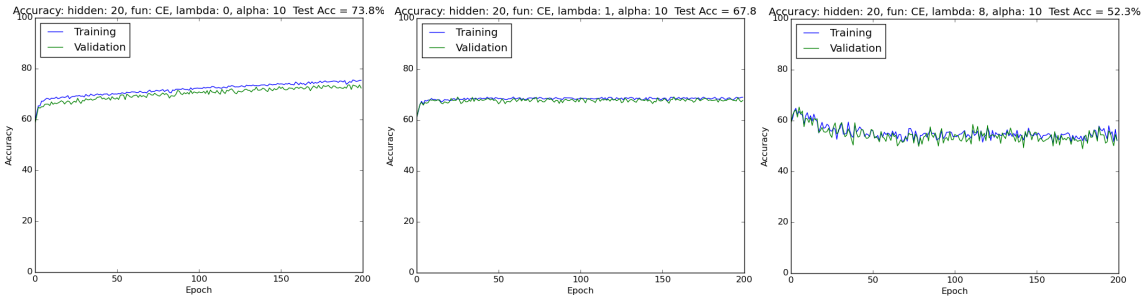

Figure 3.4: Accuracy for different values of lambda

3.3 Results for 11 instrument set, different musical notes

In this last experiment we wanted to check the performance of our system against the whole set of notes and instruments, this is the full dataset we have.

After the results of the last experiment, where the system had troubles to distinct between two instruments, we did not expect a good result. Out data, as it is explained in the appendix A, in far from being perfect. The amount of noise and the different number of samples for each instruments, make the classification a hard task. We have added this experiment in order

to show how our system fails and figure out how we could improve it to solve this problems.

In Figure **??** it can be observed the accuracy results for a system of ten hidden neurons. The test accuracy is just 31.9%, which is very far from the previous results.

The next chapter will speak largely about the problem that are causing this low accuracy and the possible future implementations that could make our system performs better.

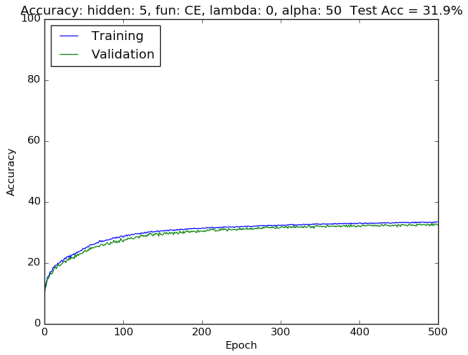

Figure 3.5: Accuracy for eleven instruments

4 | Extensions, strengths and weakness

We have develop a neural network which aim is distinct musical instruments by its sounds characteristics. In this section we will explain where our system is strong and where it finds problems, as well as possible improvements that would make the system better.

At a first step, the goal of the project was only distinct between instruments that where playing the same notes, but due to the good results obtained during this test and the difficulty for finding an extense and reliable dataset of multiple instruments playing the same notes, we increase the complexity and tried to solve a harder problem. The constraint of the same note was removed, so the classification would be applied over any instrument and any note.

As it was showed in the experimental part of the project, out network performance is high for the first approximation. Although the input audios are noisy and the musical instruments that are playing them are different (we need to remember that a C5 is very different if it is played in two different guitars due to different parameters as the construction process, the wood or even the place where the sound was recorded) the system was able to distinct them with a very low error rate.

In the other hand, our approximation is too weak for solving properly the recognition task when the note constraint is removed. This is due to the representation of the data. Harmonics are not enough in order to describe each instrument, they are very noisy when the frequency and the intensity is changed for the same instrument. More suitable representation of the data would allow us to increase the accuracy obtained, although the advantage of this is the very simple approach.

In order to improve our neural network a different and more complex layout could be use. In **??** a two-layer feed-forward neural network is applied with good results, although the goal is different to ours because the authors just want to classify the instrument in families, such strings, woodwind...

It would be also very interesting to check the efficiency of our system with another techniques such k-Nearest-Neighbours where the complexity is reduced because the only task is, given an input, look for the k most similar samples in the training and decide the instrument that is playing looking on similarities. Another option could be use Support Vector Machines, which are top approximations nowadays and allows to classify samples of very high dimensional data. This fact would allow us to use more harmonics as input in order to have more data, which is finally what makes a system work or not. An ideal thing would be find or create an optimal dataset of audios, adjusted to our main needings, where the noise is minimum and the cardinality is high.

5 | Conclusion

In this project we have dealt with a musical classification problem, where recognise different musical instruments from audio files was the final target. For this, a dataset composed of audio samples of eleven different instruments was used. The pre-processing task have been done through a Fourier Transform, in order to extract the harmonics of the wave shapes for each the instrument. These values were used as input of our system taking a total of ten harmonics for the experiments.

As concerns the classification system, we adopted a Neural Network which performs Stochastic Gradient Descent with a backpropagation step to learn from the training data and different CI techniques has been applied to the network in order to improve its performance.

Three tests has been carried on, from the most simple to the most complex. The first one using a isolate note and nine different instruments; the second one with only two instruments and the whole set of notes and a last one with all the instruments and notes of our database. The accuracy of this test is an average of multiple executions in order to ensure its reliability.

The best results have been obtained for the first case, where accuracy reaches almost the 90%. Results for the second case are worse and implies that our system suffers for differentiate instruments playing different notes. This fact is definitely probed with the last experiments where accuracy clearly drops. The reason is the representation of the data with the harmonics. They are not constant when the frequency and the intensity of the signal changes, but they vary so much. Another representation is required in order to increase the performance of the classification.

We can conclude that with the feature extraction phase that we applied, the classification of instruments that are playing different notes is not feasible. Possible solution would be an increment on the training data number, which is not easily found, and quality (lot of noise in the recordings is found). A preprocessing which include more characteristics related with other aspect of the sound as its shape, temporal features, the spread... could result in more information which would ease the classification.

Bibliography

[1] Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *Information Theory, IEEE Transactions on*, 44(2): 525–536, 1998.

[2] David Benson. *Mathematics and music*. University of Georgia. Department of Mathematics, 2004.

[3] Qian Ding and Nian Zhang. Classification of recorded musical instruments sounds based on neural networks. In *Computational Intelligence in Image and Signal Processing, 2007. CIISP 2007. IEEE Symposium on*, pages 157–162. IEEE, 2007.

[4] Antti Eronen and Anssi Klapuri. Musical instrument recognition using cepstral coefficients and temporal features. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 2, pages II753–II756. IEEE, 2000.

[5] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH*, pages 1756–1760, 2013.

[6] J. Heaton. *Introduction to Neural Networks with Java*. Heaton Research, 2008. ISBN 9781604390087. URL `https://books.google.es/books?id=Swlcw7M4uD8C`.

[7] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://goodfeli.github.io/dlbook/`.

[8] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. Lessons in neural network training: Overfitting may be harder than expected. In *AAAI/IAAI*, pages 540–545, 1997.

[9] Arie A Livshin and Xavier Rodet. Musical instrument identification in continuous recordings. In *Proc. of the 7th Int. Conf. on Digital Audio Effects*, pages 1–5, 2004.

[10] Giorgos Mazarakis, Panagiotis Tzevelekos, and Georgios Kouroupetroglou. Musical instrument recognition and classification using time encoded signal processing and fast artificial neural networks. In *Advances in Artificial Intelligence*, pages 246–255. Springer, 2006.

[11] John Moody, SJ Hanson, and RP Lippmann. The e ective number of parameters: An analysis of generalization and regularization in nonlinear learning systems.

[12] University of lowa. Electronic Music Studio. `http://theremin.music.uiowa.edu/`, 2012. [Online; accessed 22-January-2016].

[13] Esa-Pekka Salonen. Philharmonia Orchestra. `http://www.philharmonia.co.uk/explore/make_music/`. [Online; accessed 22-January-2016].

[14] VS Shelar and DG Bhalke. Musical instrument recognition using neural network. *Advances in Computational Sciences & Technology*, 5(2), 2012.

Appendices

A | Data Analysis

A.1 Acquisition

For the proposal of this project, a data set must be selected in order to train and test a neural network. We have to deal with two different points before start the analysis: choose a suitable set of data, and how to represent this data property for the input of the neural network.

Suitable data must be selected at the beginning to guarantee a good performance. The format of the data chosen in this project should be composed by audio files where a isolated instrument plays only a musical note. Therefore for each instrument there is a collection of audio files with different notes. All the audio files recollected in this project were obtained from [12] and [13]. Each audio file is named following this format: *instrument_ note_ intensity.mp3*, where the notes are decoded in the American nomenclature (**C, Cs, D, Ds, E, F, Fs, G, Gs, A, As, B**) and the intensity is given by the abbreviations **pp** (*piannisimo*), **p** (*pianno*), **mf** (*mezzoforte*), **f** (*forte*), **ff** (*fortissimo*).

However this raw data collection has to be filtered. The reasons are two: first, given this amount of files, we do not know if they are correct (i.e. if the name of an audio file is truly referring to the information it contains) and second, our implementation about computing the representation of each audio (see next section) may not work for a specific file. Therefore a filtering is required in order to avoid these possibilities and its procedure is as follows: The frequency (note) of an audio is computed from its wave by our implementation, then it is compared with the note that its name says; if the frequency coincides with the note in its name, the file is taken as valid, otherwise is invalid and we will not use it for the analysis. We can guarantee that the representation of each audio selected by this filtering will be correct and meaningful, discarding those files that have a wrong representation or name and could affect to the performance of the neural network.

Table A.1: Information of the total data recollected and the filtering

Instrument	Original audio files	Valid audio files	Filtering (%)	Lower Note	Higher Note
Banjo	74	35	47	C3	E6
Bass Clarinet	944	682	72	C2	C6
Cello	889	569	64	C2	C7
Clarinet	848	601	70	D3	C7
Flute	879	582	66	C4	F7
Guitar	112	58	51	B2	C6
Oboe	596	470	79	B3	A6
Sax	733	355	48	C3	F6
Trombone	831	392	47	E2	F6
Trumpet	485	346	71	F3	F6
Tuba	972	509	52	B0	F4
Viola	974	548	56	C3	F7
Violin	1507	935	62	G3	E8

In table A.1 is shown the information of the data recollected, such as the number of audio files for each instrument and the range of notes, from the lowest to the highest. It is also shown the filtering process

Once the data is recollected and filtered, we can consider the data acquisition process as done. The second step is find a representation of this data, in order to introduce it to the neural network.

A.2 Representation

Data representation is perhaps the most important part of the data analysis, where all the samples are represented with a set of meaningful features. This is taking into account that an audio file contains a lot of information about the sound recorded, so we need a suitable way for represent this information with a few values. These features will be the normalized in the range $0, 1$ and inserted into the first layer of the neural network.

Many different ways of representing the audio files of the instruments are documented and implemented, being more or less complex: for instance an easy way is choose the *Duration* and *Shape* of positive repeated wave fragments [10]; or by choosing more features such as *Temporal*, *Energy*, *Spectral*, *Harmonic* and *Perceptual* features [9], or features related with the with the

frequency (*Brightness*, *Spread*, etc) [4].

The chosen representation in this project is the **Harmonic** analysis. The reason is because Harmonics can represent very well the shape of a wave, and its implementation is really simple. This also gives us flexibility with the number of Harmonics, related with the number of input neurons. The **Harmonics** of a periodic wave are the coefficients of the Fourier Series of the wave. Mathematically, a periodic function can be described with a Fourier Series in the following way: Let $f(x)$ be periodic with period T, then the Fourier Series is given by the expression in A.1

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} \left(a_n cos\frac{n\pi x}{T} + b_n sin\frac{n\pi x}{T} \right) \tag{A.1}$$

The coefficients a_0 and b_0 can be added together in a compact way $h_n = \sqrt{a_n^2 + b_n^2}$. We call these coefficients h_n the harmonics of $f(x)$. The expression to compute these coefficients is shown in A.2:

$$a_n = \frac{1}{T}\int_0^T f(x) cos\frac{n\pi x}{T}\,\mathrm{d}x \quad b_n = \frac{1}{T}\int_0^T f(x) sin\frac{n\pi x}{T}\,\mathrm{d}x \tag{A.2}$$

Therefore given the period signal of an instrument, we represent the whole signal with the harmonics h_n (For more detailed information, see [2]).

We have implemented a way of computing the period signal of an audio file, and therefore its frequency. However this method is not correct from all the audio files recollected. We tried to improve it in order extract the correct frequency and period for all the files, although some of them have weird wave shapes and it this task is not possible. This is the main reason of the filtering step we have previously done: only the files which frequency, extracted by our implementation, is right are taken. Imagine that the frequency (and then the period) extracted is wrong: since the harmonics are computed within the period interval, these values will not be representative for the audio file.

In figure A.1 is shown the $A3$ note for the instruments *Clarinet*, *Bass-Clarinet*, *trumpet* and *Viola*. We can appreciate how different the shape of the waves are, and that they have the same period. We now compute their Harmonics in order to represent them. The results of the harmonics are shown in figure A.2. In the left of each plot is plotted the original signal and the synthetic signal computed from the Fourier Series. In the right are shown the harmonics computed from the coefficients. This approach has be done with 10 harmonics. Higher number of harmonics can be taken in order

to increase the fit of the wave, but we will work with this number since the input number neurons in the neural network will be 10 (and higher input neurons will result in more computational time) and the fit of the Fourier Serie is actually very good.

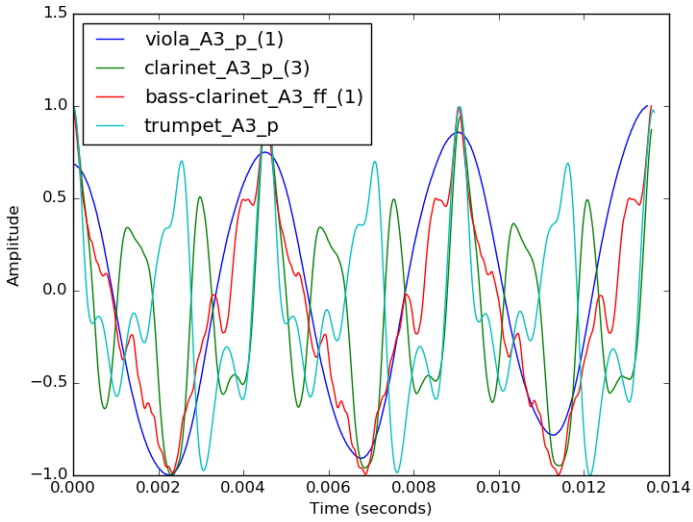

Figure A.1: Signal of 4 instruments playing the same note

Once we have described the representation of each audio file, we wonder if this representation would be similar for the same instrument. This is required in order to classify the instruments, since we need similar representation for the same instrument for a good performance. In figure A.3 we can see different signals for the same instrument Viola for the same note. Observe that there are differences in the signal, depending on the intensity of the playing. In A.4 are shown also the harmonics of these four signals. The harmonics have the same behaviour although their values vary for each file. This is not the only problem, since the wave shape of an instrument changes, depending on the note and the intensity. With this representation of the data, we find this inconvenient but on the other hand it is very simple. We will see the performance and results of the neural network using this representation, and conclude if the representation is suitable or not.

A.3 Pre-processing

Once we have described how to represent the data and its inconveniences, we will train a neural network in order to classify some instruments from audio

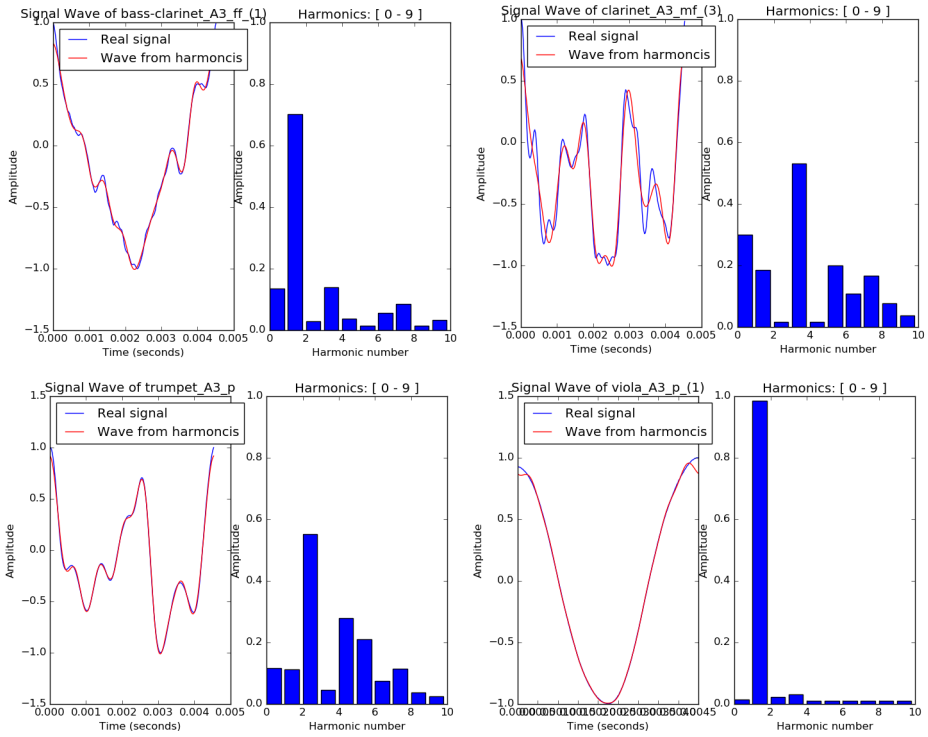

Figure A.2: Harmonics of the the 4 instruments shown in figure A.1

files in format "*.mp3*". First, a data set is prepared with a fixed number of instruments, including audios with all the possible notes (We will try to classify the instruments no matter what note is playing). The pre-processing of this set of audios is done by computing the harmonics (fixed to 10) and the labels (the instrument name) from the name of the file. Second, given the number of different instrument, we construct the expected output of the neural network from the labels. This output will be an array with its length being the number of different instruments. The array will contain all 0 values except in the position of the array corresponding which each instrument, where there will be a 1. For instance, if we want to classify three instruments *viola, flute, trumpet*, the corresponding output of the neural network will be $1, 0, 0$ for the Viola, $0, 1, 0$ for the Flute and $0, 0, 1$ for the Trumpet. Finally the training of the neural network will be done using the harmonics as the input parameters (setting first the number of input neurons as the number of harmonics) and the expected output for the training (setting fist the number of output neurons as the number of different instruments).

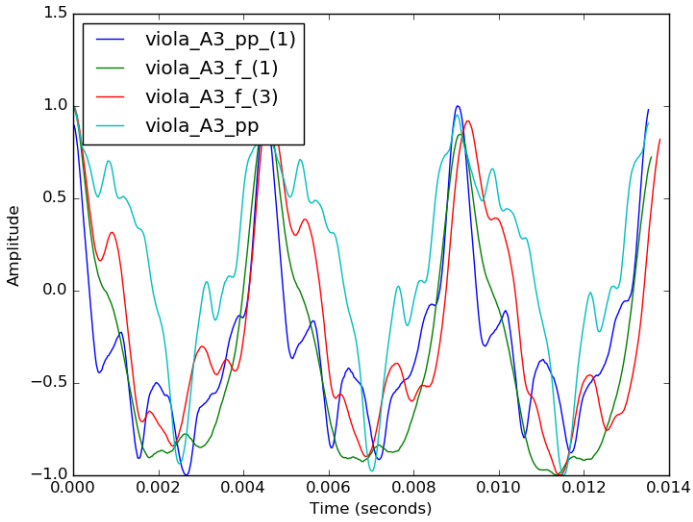

Figure A.3: Signal of 4 different records of the instrument Viola

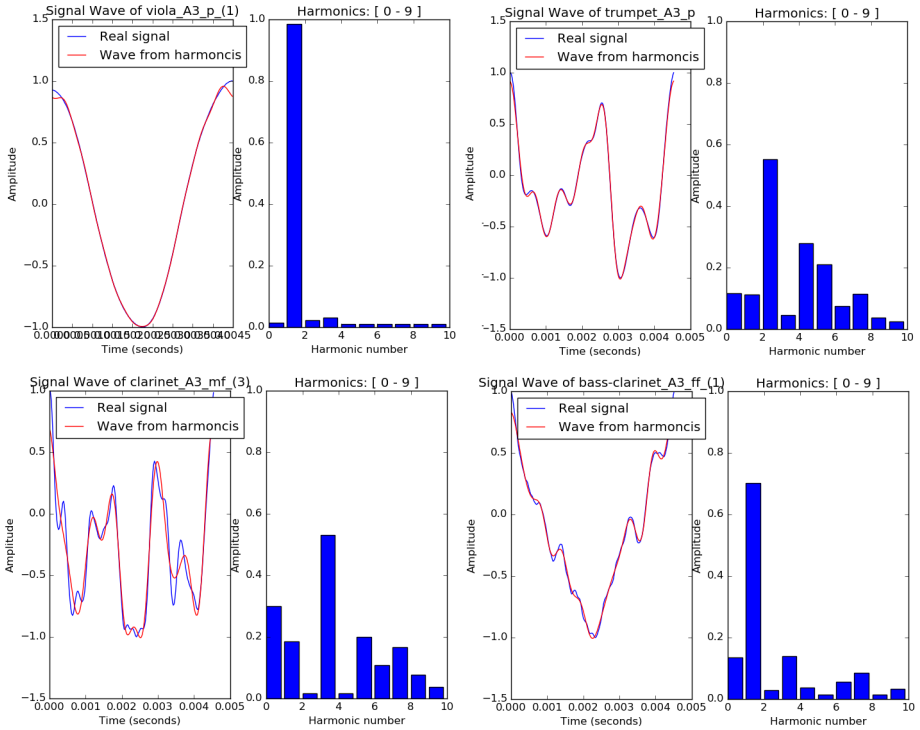

Figure A.4: Harmonics of the the 4 signals shown in figure A.4 for the instrument Viola

B | Implementation details

B.1 Network Development

At a first step we thought in use TensorFlow, Google's AI engine, but we decide in develop our own implementation of the network in order to completely understand how backpropagation algorithm works. For this we choose Python as our programming language due to its simplicity and the facility of finding information in the web. The neural network code can be found in "nn.py" file, which is a python class which methods implement the different points of the algorithm, such the activation and cost functions. The most important one is the backpropagation method which allows us to train the system using a previously labeled dataset.
The concepts of the algorithm will not be explained in this document due to they are included in the topics of the subject, although we will include explanation of the different improvements we applied over the basic implementation. The development has been carried on through multiple iterations which increase the network functionality.

B.1.1 A first approach

In this first step, the basic system was deployed. This is neural network with three different layers and the sigmoid as the activation function of every neuron. We decide to set the weights and bias according to a Gaussian distribution with mean zero and standard deviation one. For sure that there are better ways of initializing this parameters, but the system works well with this set up so we will let this problem as a future improvement. The selected cost function for this first approach was the classic mean squared error. In order to test that the implementation is correct a basic script was created ("xor.py"). The idea behind this is make our network to learn the classic Xor function, with just two digits. Thanks to this, the execution is very fast and we can check and debug our code easily. The performance of this is showed in Figure C.5, which shows a mean of the cost function over

100 iteration vs the number of epochs.

B.1.2 Cross-entropy function

In the previous approximation we use Minimum Squared Error (MSE) as our cost function, because it is simple to understand and implement. Although this simplicity also brings some problems that can be properly solved using another function as cost. What we are doing is a multi-class classification, and it is well known that for this kind of goal a Cross-Entropy function (CE) will perform better. Golik et al. [5] have done a deep analysis of both functions in an experimental an theoretical comparison. Their experiments have shown that in a comparable environment with randomly initialized weights, CE allows to find a better global optimum. It is worth it to note that, as we are using the sigmoid as activation function, an important fact that is affecting the learning of our system. When we compute the backprogation step of the algorithm, the cost derivative is computed with respect to the weights and to the biases. These derivatives involve the derivative of the different activation functions during the backwards path. In a very brief summary the backward process consists in products between the cost derivative and the sigmoid derivative (see equationB.1).

$$\delta_j^{(l)} \equiv \frac{\partial E_{emp}^{(n)}(\mathcal{W})}{\partial a_j^{(l)}} \tag{B.1}$$

If we have a look on Figure reffig:sigmoid, the sigmoid function derivative is practically zero when the output is near to 0 or 1. LetâĂŹs think in a situation where the output of the neuron and the expected output are very similar. The cost function will return a small value, as well as the sigmoid derivative. For this reason, the weights and bias changes will be very small (without considering the learning rate). Nevertheless, when the results are totally wrong, this is the an output and expected very different, the cost function derivative will be also small because of the sigmoid derivative value. So the weights and bias modifications in the gradient direction will be small, we will need a lot of epochs in order to improve the cost. Hence, using the cost function will result in a slow learning when we are far from the solution and as we are using a Gaussian distribution for the initial parameters, this fact can easily happen.

We add a parameter to the class constructor which allows us to select between the two cost functions, in order to facilitate the testing and the

results retrieval. The cross-entropy for our multi-layer output is the following:

$$E_{emp}(\mathcal{W}) = -\frac{1}{n}\sum_{n=1}^{N}\sum_{k=1}^{m}[y_{nk}\ln a_j^L + (1-y_{nk})\ln(1-a_j^L)] \tag{B.2}$$

Where $\mathcal{W}$ is the set of all network weights, N is the size of the dataset, k is the number of attributes per sample and a^L is the output of the neurons in the last layer. This notation is based on the course slides and in [7]. We need to compute the derivative of this function in order to apply gradient descent method and backpropagation. Our new set of equations will be:

$$\delta^L = \frac{\partial E_{emp}(\mathcal{W})}{\partial a_j^L} = a^L - y \tag{B.3}$$

$$\frac{\partial E_{emp}(\mathcal{W})}{\partial w_{ji}^L} = \delta_j^L \cdot \frac{\partial a_j^L}{\partial w_{ji}^L} = \frac{1}{n}\sum_{n=1}^{N}\sum_{k=1}^{m} a_i^{L-1}(a_k^L - y_k) \tag{B.4}$$

As we can see, the derivative of the sigmoid is vanished and then it is not affecting our backpropagation algorithm. Thanks to this, we can solve the previously commented problem, where the neurons suffer from a low learning when the cost function value was big. The rest of the backpropagation method stays the same. The implementation of this code can be found in "nn.py" file, one of the parameters that must be passed to the constructor of the class, is the chosen cost function. We use "MSE" for the classic minimum squared error cost and "CE" to denote the cross entropy. It is possible to change this cost after creating the neural network, in order compare them. This implementation has also been tested with the xor script to ensure its good performance.

B.1.3 Regularization

One of the hardest things in this project has been finding a valid set of training data. There are not many free datasets with the audio information we were looking for. For this reason the use of regularization is totally necessary. As we have been during the course, regularization is one of the most known techniques for reducing overfitting and its performance has been more than proved [1]and [11]. The proposed regularization technique is the classic $L2$ norm, which also was covered in the topics of the subject and for this reason we will give just a short description of the method.
The basic idea of regularization is add an extra term to the cost function, which takes into account the value of the weights in the cost function. This is that we will not allow to the weights be of any side, the constraint added

to the cost its the squared sum over all the weights. Since now on we will only take into account the cross entropy cost in our report.The resulting cost function will be:

$$E_{emp}(\mathcal{W}) = -\frac{1}{n}\sum_{n=1}^{N}\sum_{k=1}^{m}[y_{nk}\ln a_j^L + (1-y_{nk})\ln(1-a_j^L)] + \frac{\lambda}{2n}\sum_i\sum_j \omega_{ij}^2 \tag{B.5}$$

We need to add this term to our backpropagation algorithm, and it is a very simple thing, because with the followed approach just a few lines of code are needed. Two things were necessary, add the the regularization term to the *costFunction* method and add its derivative, which is self explanatory, to the weights update.
It has been added a new attribute to the NN class, which is the lambda parameter. When it is set with a value distinct from zero, the neural network will take into account the regularization term in its training.

B.1.4 Softmax

At this moment, cross-entropy is our cost function because it allows us to solve the problem of the neuron saturation related with the sigmoid derivative. Since the first moment we said that we will use the sigmoid as activation function for all the neurons, but it is time to add a new function to our system.
We are developing a classifier network, where the last layer has as many neurons as instrument we want to classify. Each neuron corresponds with one instrument and it is activated when the input are the harmonics of an audio played by that instrument, in a certain way we can say that each output neuron is mapped to an instrument. In an ideal situation, the output of the network would be a vector with only a one, corresponding with the one which recognise the instrument. But normally this is not happening, it is worth it to noise that as we are working with audio sample of real instruments, noise is affecting our system. This, together with another factors will result in an output vector where more than one value is distinct to zero.
This is the reason of using *softmax* as activation function in the last layer, which takes into account all the neurons at the same time.

$$a_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{B.6}$$

As it happen with the sigmoid, the combination of cross-validation and *softmax*, (taking into account that we are using a classification network), allows

us to say that:

$$\frac{\partial E}{\partial w} = a_i - y_i \tag{B.7}$$

Because we can consider that only one neuron activation will be different from zero

C | Figures

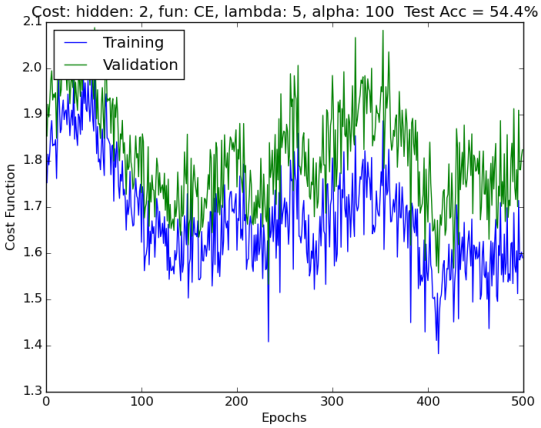

Figure C.1: Accuracy $\alpha = 100$ and two hidden neurons

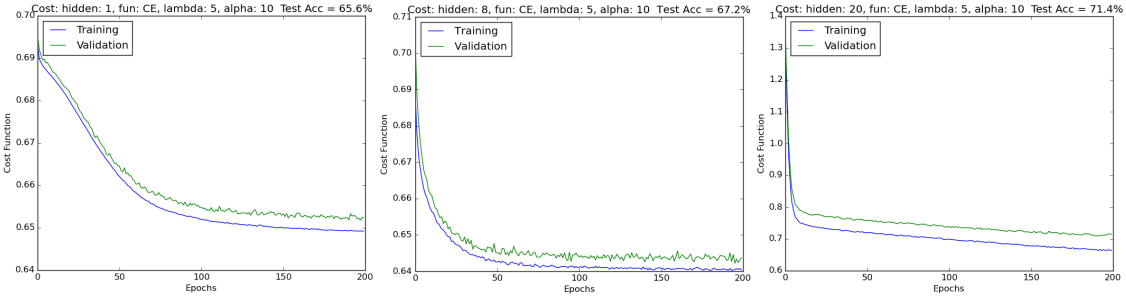

Figure C.2: Cost for different numbers of hidden neurons

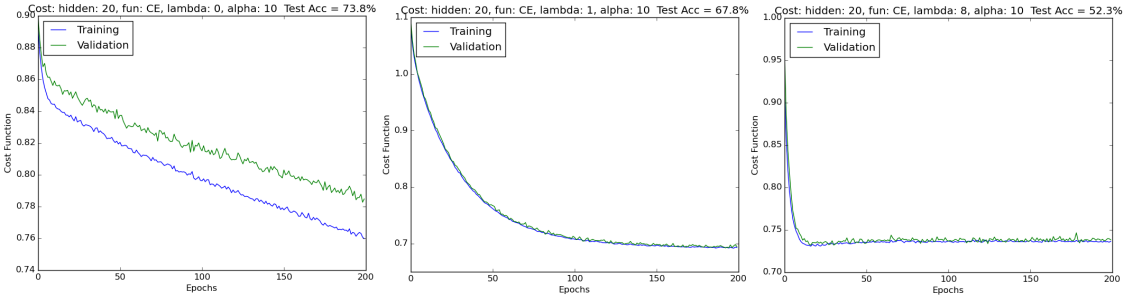

Figure C.3: Cost for different lambda values

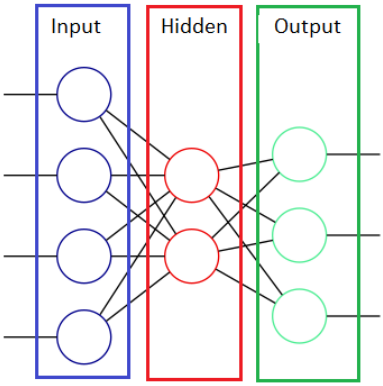

Figure C.4: Classical Neural Network layout

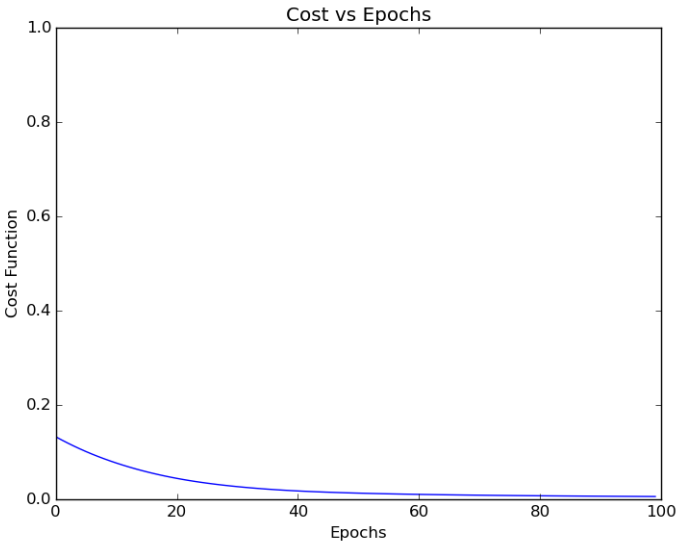

Figure C.5: Epochs vs Cost averaged results after 100 iterations

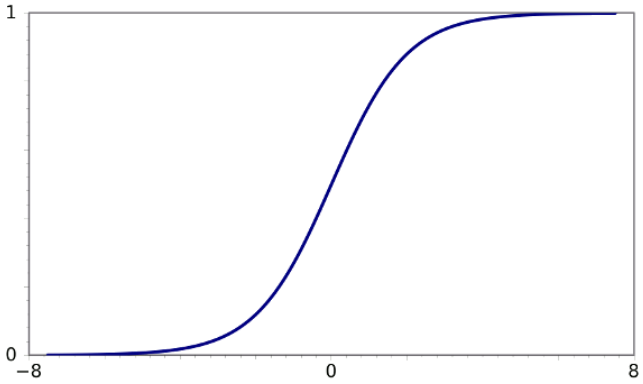

Figure C.6: Sigmoid function

D | Tables

Hidden neuron number	Batch size	Lambda	Alpha	Accuracy
5	1	0	1	62.0%
			10	83.8%
			50	82.3%
		1	1	13.2%
			10	9.1%
			50	4.5%
		3	1	12.4%
			10	11.1%
			50	11.2%
	20	0	1	64.2%
			10	82.6%
			50	81.0%
		1	1	60.4%
			10	62.9%
			50	38.0%
		3	1	46.7%
			10	47.2%
			50	19.1%
	121	0	1	63.7%
			10	77.3%
			50	9.1%
		1	1	67.5%
			10	74.8%
			50	11.1%
		3	1	60.8%
			10	71.5%
			50	13.6%
10	1	0	1	75.2%
			10	86.0%
			50	89.7%
		1	1	9.9%
			10	13.2%
			50	11.1%
		3	1	9.9%
			10	9.9%
			50	12.0%
	20	0	1	70.7%
			10	85.6%
			50	85.9%
		1	1	65.8%
			10	64.9%
			50	47.1%
		3	1	48.4%
			10	43.4%
			50	12.0%
	121	0	1	69.0%
			10	78.1%
			50	12.0%
		1	1	67.8%
			10	77.6%
			50	12.0%
		3	1	67.8%
			10	77.6%
			50	6.2%

Table D.1: Results of the set: 9 instruments playing the note A