Introduction to Neural Networks with Keras

Florent Martin

DON'T FORGET TO RUN THE FIRST CELL OF THE NOTEBOOK

Hello everyone First I would like to thank the orgaizers of PyData Munich and super Python talks and in particular Nick del Grosso who got much involved in the last weeks and help me to improve the quality of this presentation.

Before starting I would like to encourage all of you to interupt me and ask a question something is not clear during this talk, or if you have a remark.

Introduction

So the goal of the 45 minutes is to introduce neural networks and see how to use them with the Python library Keras.

First an overall introduction:

What is machine learning? It is a field of computer where one solves a given task, and instaed of developping a classical algorithm to solve this task, one chooses a model. And the important thing is that the model depends on some parameters (in most cases those parameters are numbers). And to use the model, you need to fix the parameters of the model. To do that, the strategy of machine learning is to use some data (often this means: instances of the problem with their solution) and after seeing all the data, your model chooses the best parameters that fit the data.

Now, neural networks are just a family ofmodels which are used in machine learning. And I want to say that neural networks are just one example of models for machine learning but there are many others models (for instance Decision trees, support vector machines, naive Bayes classifiers and so on).

And Keras is a Python library which allows to use neural networks.

So the goal of this presentation is to solve a simple problem (flower classification) using Keras and try to understand how this works.

Before starting I want to anticipate a make a sketchy overall of what we'll see. Here on the top right you see a graphical representation of a neural network. The idea is that you've an input (which is a number) and you want an output (which is also a number) and to make the computation of the output there are some intermediate nodes. One calls this layers in the middle some hidden layers (in the sense that they are not part of the input nor of the output) but still are crucial in the computation.

And I should say that the parameters are used in the intermediate nodes. This means that each node in this computational graph depends on some parameters. On the left side, what you see is a neural network which is simpler and this very kind of neural network have there own theory, and their own names: namely they are called logistic regression classifiers. So actually in this talk, we'll first investigate logistic regression (because it's simpler to understand) and then we'll move to more complicated neural networks like on the right. And the curve you see on the left, is the typical kind of curve a logistic regression classifier can learn. And the curve on the right as we'll see t is a typical example of a curve that can't be learnt from a logistic regression classifier but can be learnt by a general neural network like this one.

The plan will be as follow: there will be 3 parts. The first part will be about logistic regression. First we'll investigate a problem of flower classificatin, for that we'll use the socalled iris dataset. Then we'll implement our first logistic regression using scikit-learn which is a very general library to do machine learning with python. And we'll finally use Keras to implement a logistic regression.

In the second part we'll talk about gradient descent which is the technique that neural networks use to fit the parmaeters of the model with the data. For that we'll talk about optimization, and then about cross entropy loss function.

And in the third and last part, we'll implement with Keras a neural network with a hidden layer like we saw on the previous slide. For that we'll change a little bit the problem to solve, then try to use logistic regression, but it won't work we'll try to understand why and what are the limitations of logistic regression, and finally we'll be happy to see that neural network with a hidden can solve our problem.

1 Before Neural Networks: Logistic Regression

1.1 Iris dataset

Before starting, I will import the python libraries we will need. Numpy which is used to manipulate arrays of numbers and do some numerical computations, tehn Matplotlib which is a library to plot 2-dimensional graphs, then pandas which allows to manipulate datasets and finally seaborn which is a vizualisation library which we use on top of matplotlib and allows to improve the quality of the plots.

Let's now start the first part about logistic regression. As you'll see, logistic regression is an example among many others of neural networks, and one could say that logistic regression is the simplest neural network one can build. And I believe that before seeing the general notion of neural networks, it is better to get a first contact with logistic regression.

For that we'll ivestigate a flower classification problem and use a famous data set called the iris dataset.

It can be loaded directly with seaborn using the fonction load_dataset. Now the variable iris is a dataframe and we can see 5 random rows of it using the method sample of the dataframe iris.

Each row in the dataframe is a sample of an iris. We have 5 columns : sepal length, sepal width, petal length and petal width and a last colums that gives the species of the sample corresponding to the row. We can see the number of rows and colums of the datframe using its method shape: it has 150 rows (which means for us 150 samples) and 5 columns (which we had already seen above). And we can see how species are distributed in this dataset usnig the method value_counts on the column species, and we see that there are in total 3 species : setosa, versicolor and virginic and that we have 50 samples of each.

In case you wonder what an iris is, or what a setoss, a versicolor or a virginica is, here are pictures of each of these species.

With seaborn we can draw a pairplot of the dataset, where in blue you have setosa, in green versicolor and in red virginica. On the diagonal, we have histograms of the 4 properties (petal widt, petal length, and so on) and out of the diagonal, we have scatter plots between two of the features (for instance a scatter plot of the petal length verus petal width).

Problem we'll try to solve is the following: if a give you the petal width of a sample, I want to know if the sample is of the species virginica. In other words, the input is the petal width of an iris sample. The question is: is the sample's species virginica or not? Hence, the ouput an answer to the question: is the plant a virginica? The anser could be yes (or 1 if we want a number representation) or the answer could be False (or equivalently 0).

To work on this we'll ad a column called 'is_virginicia' to our dataset, and the value of the column will be 0 if the plant is not a virginica and 1 if it is a virginca. And we can call the method sample and see 5 random samples and see the new column 'is_virginica'.

To answer that question, it might be good to plot a histogram of the petal width To do that we can use the method hist (for histogram) of a groupby object. To make the histogram nice, we add a legend which will plot the bins of the histogram with different colors for the species depending on wether the species is virginica or not, and we label the x-axis with the 'petal width', and we label the y-axis number of sample. The histogram we get is the petal width on the x-axis, , on the y-axis, how many samples in our dataset have this petalwidth, and in green we have virginica samples while in blue we have non virginica.

What's clear from this histogram is that on the right side, we have green bins, hence the species of the samples is virginica, while on the left side, we have blue bins, samples are not virginica More precisely, if we want to answer our problem, namely answering the question 'is the iris sample of the species virginica, for values lower than 1.3, the answer is False/no/0, and when the petal width is greater than 1.8 the answer is yes it is virginica, so the output is True or equivalently 1.

So just from this diagram, we can start trying to solve our problem, and we'll draw our answer using a dotted red curve. From the analysis we already made for petal width smaller than 1.3 the anser is 0 (keep in mind that 0 = False) and for petal width values greater than 1.7, the anser is True which we numerically represent with 1).

But in between it's not so clear. For instance when the petal width is 1.5 there are as many samples which are virginica than not virginica. So I would say that there is no obvious answer for such values. So one natural

possibility here would be to draw a line to join these two partial dotted red curves we already have. We would get that for instance.

That looks like a reasonable guess, but could also have tried to make something smoother like that: with this solution to the problem, the output is again clearly 0 so false on left, and 1 so true on the right but the difference is that the transition is very smooth.

What we did so far was trying to solve the problem using our intuition, so now let's do this using Logistic regression has I promised it earlier.

For the problem we are conidering, the output of the logistic regression classifier will be a function having the petal width as input and returning as output a probability which is the probability the the iris sample is a virginica. And I want to stress here that a probability is a number contained between 0 and 1 and the closer it is to 0, the more the prediction is close to False, and the closer it is to 1, the closer the prediction is to be True. Ultimaltely, the probability is used to answer the problem by saying: if the probability is greater then 0.5, then return True, if the probability is less than 0.5, return False.

To model probabilities the key ingredient of logistic regression is the sigmoid function. It is the function that takes as input x and returns 1 divided by 1 + exponential of -x This might be a little bit obscure for the moment, so let's plot the sigmoid function. I define a Python function the computes the sigmoid function and we call this function: sigmoid. And we plot the sigmoid function

Before when we've been trying to guess dotted red curve for the probability of being virginica, we got similar curves. Let me highlight some key properties of the curve of the sigmoid function. It takes values between 0 and 1 (that's we want because we want to use it to return a probability and as we said before a probability should be between 0 and 1). It reaches the value 0.5 (one half exactly for the value x=0) and the more you go on the right, the more you tend, and the more you go on the left, the more you tend to 0.

So the idea of logistic regression is to use the sigmoid function to output the probability knowing the value of the petal width. So let's replot the histogram of the petal widths and plot the sigmoid on the same graph.

Hum, what did we expect? The sigmoid curve is the same as before and namely it predicts high probability (higher than 0.5) for positive values. So if we were using the sigmoid red curve to predict the probabilities of being virginica, we would answer True for all positive values, and since of course, the petal widths are always positive values, we would always answer True. This is quite bad since there are lots of plants which are not virginica.

That's where the key idea to introduce parameters enters in the game. Namely we are going to modify the sigmoid function using two parameters a weigth W and we first give it the value 1 and a bias B and we first give it the value 0 and we modify the function to compute the probability replacing x by w*x + b If I plot again, I get the same thing, because w =1 b = 0 hence w times x + b equals 1*x + 0 equals x

But let's change w and make w= 5 for instance. Can someone tell me what's going happen?

the curve is compressed around x=0. It's as if we had zoomed in on the red curve. In particular the zone where the probabilities change from 0 to 1 is much smaller this probability curve is giving which confident predicitons.

Let's go back to our initial values, that is w = 1 and b= 0.

Now what happen if I set w to 3? Has someone an idea? it shiffts the curve to the left by 3 units to be sure, i set w to -3 it should shift the curve to the right yes let's set w to 1 b =0 to get the genuine sigmoid function

At this point I'd like to pause a little and come back to what I said earlier. Remember that we said that in machine learning one uses a model which depends on some parameters to solve a given task. Her the task is to output a probability that an iris sample is virginica, knowing as input its petal width which is denoted by x. Our model to compute the probability is the function sigmoid of w times x + b. So the parameters of the model are w and b. And as we saw in the last minutes, if you change the parameters, so if you change w and b, it will change the solution to the problem. In particular, what we want to find are parameters w and b that fit the data we have. So let's try to make it feet with the data we have: namely the output should be yes (close to 1) for values higher than 2 and close to 0 for values lower than 1.5 set w=3 to get a curve we a important slope and now we want to shift the curve to the right, so we need a negative value for b. try b = -1 , not enough try b = -10 too much try b= -5 OK

Let's sum up what we saw from logistic regression: it is a model which depends on 2 parameters W and B and it ouputs a probability sigmoid(Wx + b)

So fine but in practice, you want your computer to do the job for you and guess himself the parameters.

1.2 Logistic Regression with scikit-learn

For that we're gonna use first the python library scikit learn which is a library that implements many machine learning models.

We import the class Logistic regression from scikit learn and we instatiate an instance of a logistic regression classifier and save it to the variable model.

But at that point, the model has not seen any data. And the philosophy from machine learning is that the model should adapt to the data we have, and it should learn and modify its parameters so that they fit the data. To do that, you have to call the fit method from the model, and give him the input data and the output in our case, the output is 0 if the plant is not virginica and 1 if the plant is virginica.

Now the model has read the data, and hopefully now the parameters fit the data

We can use the attributes coef which represents the parameter w and intercept which is the parameter b.

And it is quite close to what we had found ourselves before: we had by hand playing with the curve guessed w=3 b =-5

So let's try to plot the probability curve for that we use the method predict_proba from our model and we've got this curve, this really agrees with what we've seen so far, it looks like the sigmiod graph shifted to the right

Let's plot on top of it the histogram we had And indeed the predicted probabilities from our logistic regression model which are plotted in red are very satisfying (at least very close to what we had suggested before)

Now an important question is how to know if your model is doing going on the data? Here of course it is easy because the input has only one coordinate, but if there were many inputs it could be difficult to evaluate the model.

For that one introduces the accuracy of the model : it is the ratio between number of samples in the data which have been correctly classified and the total number of samples

with scikit-learn, the method score from the model returns the accuracy and you need to give as input to this method the input values of the dataset (in our case the petal widths) and the outuput (1 if the plant is virginica, 0 otherwise)

1.3 Logistic Regression with Keras

Let us now implement a logistic regression classifier, but this time using Keras. For that we import the class Sequential from the module keras.models, and we import classes Dense and activation from the module keras.layers.

If you remeber, with scikit learn we just asked to python please create an instance of a logistic classifier, and we didin't have to ask more. In particular, the knowledge that under the hood the model computes the function sigmoid of w times x plus b was not necessary at all. With Keras it is different, and the reason is that Keras uses neural networks as models, and there are many different sorts of neural networks, or let's say there are many different sorts of architectures of neural networks, and logistic regression happens to be the simplest architecture of a neural network. To understand how keras works and what neural networks are, it is a good idea to repreent the computation underlying logistic regression with a computational graph.

On this computational graph, the computation of the function sigmoid of w times x + b is seen as sequence of elementary operations. And in the terminology of neural networks which is also the terminology of Keras, we call each step in this sequential representation a layer. So here the first we start our sequential computation with the input x, then there is a first layer of computation where the input x is transformed into the value w times x + b, and the last layer in our sequential computation is to apply the sigmoid function to wx+b.

And this sequential representation of the computation in terms of layers which represent each step of the sequence is exactly the way you define your model with Keras. Remeber we imported a class Sequential from Keras and this will be the class of our model. So now the variable model, which is an instance of the class Sequential represents our underlying computational model, and the constructor for the class sequential is given a list of the steps in the sequence of computation, and remember I just said that these computational steps are called layer in Keras. Here the first layer is called a Dense layer. Since it is the first layer in our sequential computation, we need to say to Keras what is the dimension of the input. In our case we have just one variable, hence the input_dimension is 1. And in Keras, the so called Dense layers, they compute functions of the type x is mapped to w times x + b (in mahtematical terminology, dense layers compute affine functions) and specify the second step of our sequential computation by adding an activiation layer of type sigmoid which will apply the function sigmoid to the previous value in the sequential computation, namely it applies sigmoid to the value wx + b.

With Keras need an extra step, namely you need to compile the model (we'll come to that later) Then we can proceed like with scikit-learn. Namely, we fit the model with the data we have (and we'll come later to the extra parameters epochs and verbose. And once the model has fitted the paramters, we can get them using the method

get_weights of our model. To evaluate the model, it's different than with scikit-learn: first just to show you I print the metric_names attribute of the models and it tells me that there are 2 metrics to evaluate the model: one called loss, and one called acc for accuracy. And to evaluate the model we call the method evaluate of the model using the input data and the output data, and we get 2 values the first is the loss (I'll explain about it very soon) and the accuracy. Here we get the same accuracy as before, namely 0.96.

Finaly, we can plot the curve of the probabilities predicted by our keras model. For this we use the method predict from our model. And we plot also the histogram, and we get the same result as with scikitlearn.

So thats it for the first part where we saw the model behind logistic resgression, and we saw how to implement it with Python with scikit-learn and with Keras.

2 Gradient descent

So now we start with the second part about gradient descent. So gradient descent is the answer to the question: how does scikitlearn or keras finds itself the parameters of the model? In other words, how does the method fit works (remember that both with scikit-learn and Keras, our models had a method fit that we had to call in order for the computer to compute the parameters of logistic regression model.

2.1 Optimization

For that we first need to talk about optimization. First let us create a logistic regressio model with Keras and fit with our dataset. So I create an instance of Sequential, I add the 2 layers: the dense layer and the activation sigmoid layer, I need to compile the model, and I fit it with the data, but remeber before I used the keyword verbose to put hte vebose mode off, and I don't do it this time, and what happens is that Keras prints a quite long log list. What happens is that Keras starts with some random paramaeters: it chooses random values for the parameters w and b, and then tries to modify w and b to improve them in the sense it tries to make them fit better with the dataset we've given to it. And this improvement of the parmaeters is made step by step, and each step is called an epoch. So for instance, when we specified the keyword epochs = 500, that meant we said to Keras make 500 steps of improvement of the parameters. And by default keras reports in this log list for each of the epochs. Kers tells us that the first epoch took it 800 micro seconds The 1/500 says the Keras reports on the first epoch out of 500 hundred as we asked, here 150/150 means that in one epoch Keras has been using all of the samples and recall that there are 150 samples of iris. And the most important for us is that Keras outputs 2 values: the loss and the accuracy. So we already talked about the accuracy: it is the ratio of correctly predicted species from our model. So for instance, it says that at the end of the 1st epoch the accuracy is of 0.33 So let's have a look to what happens to the accuracy when more ond more epochs are made. We see that it improves epoch after epochs.

So that's an important idea to have in mind: what Keras or scikitlearn actually does when we call the fit method, it is making a step by step improvement of the parameters to increase the performance of our model. In more technical terms, Keras is trying to maximize its score. By the way, this is kind of what we did ourselves in the first part when we played wiht the paramters by hand. I don't know if you remember, we modified the value of the parameters w and b and at some point we wanted to shift the red curve to the right, for that, we decreased the paramter b, but it was first not enough, then we changed again but then it was to much, and finally again modified b and we were happy with how the red curved was fitting the histogram, so the data. And it seems that this is what Keras is doing. So let's try to understand how Keras is doing this.

From what we'v seen so far, a reasonable guess would be the following: the accuracy of the model seems to be a easy to understand a good score to evaluate the model, so maybe, what Keras is doing, is trying at each epoch or step to improve the accuracy of the model on the dataset.

To see how reasonable this, we're gonna plot with a heatmap the accuracy according to parameters. For that we define a meshgrid and the idea is that on the x axis there will be the paramter w and on the y axis the parameter b. Then we define a function called get_accuracy which takes as inputs the parameters w and b and returns the accuracy when the model has the parameters we gave to it. And finally we call vecorize_accuracy a vectorized form of this function, this will just allow us to easily give arrays as inputs. And we call the function vectorize_accuracy on our x grid and y grid and stores it in acc_grid, so we have every thing to plot a heatamp of the accuracy.

So what we have here is on the x -axis the parameter w which takes values between -10 and 10 on the y axis we have the parameter b with also values between -10 and 10 and the colors represents the accuracy, and according to the colorbar on the right, red means an ccuracy close to 1, blue means an accuracy close to 0 and white/pink means accuracy close to 0.5.

Now on top of that, we want to see how the model evolved epochs after epochs. For that I first specify some initial parameters values for the model and make a for loop where in each loop we run 25 epochs of the fit method and we print a block cross on our plot to see where is the model.

So first the model was in the the left-bottom part this was my intention make it start there (remember I set the starting parameters with values -9 and -9). And here we see that after each steps (the difference between 2 corsses is 25 epochs) we really see the model going to the red zone where the accuracy is higher.

But the strange thing here, is that the accuracy is constant in the wole left bottom zone. The accuracy there is literally constant. And I forced Keras to start its fitting process with parameter values in the middle of this pink zone. I don't know if you feel this but if you have in mind that your model should try to update its parameters w and b in order to improve the accuracy, this was a tricky starting point, and more importantly, the behaviour of keras that we see with this black curve (and let me say again that the black curve represents the parameters of the model at various steps of the fitting process) tends to indicate the Keras was not really trying to improve accuracy because it stayed for a very long time in the pink zone. So probably there must be something else that keras was considering.

2.2 Loss function

Indeed there is something else which is called the loss function. I want to warn you the next 2 slides will a little bit technical. So the loss function is defined as the some over all samples (samples are represented with the variable i) In this formula y_i is the actual output of our sample: for us it is 1 if the sample is virginica and 0 if not and p_i is the probability that your model outputs. Then the loss is the sum over all samples of y times log p + 1-y times log p. Actually this formula looks very obscure when you see it for the first time. But let's recall that y takes only 2 values 1 or 0 and the formula can be simplified if you have this in mind. When y is y log p + (1-y) log (1-p) simplifies a lot because 1-y becomes 0 and then we have just log p remaining. Likewise when y=0 this expressions simplifies to log(1-p)

When the sample has is a virginica if you predict a output probability of 1 the loss will be 0 while when the output probability is closer to 0, then the loss increases. So the idea here is that the loss has to be thought of as a penalty of the probability the model output.

Here one could make an analogy with a grade to an exam: the accuracy is the analogue of grading a student by saying either you pass, either you don't pass the exam. and of course it might be hard for the student what was going wrong with its solution. On the other hand the loss function that we see here is quite like the grading german system where the best grade is 0 and the higher the score, the worse you did at the exam.

So this loss function that we'e just defined, happens to be the loss value that keras outputs.

So now we can try to plot the same graph the we did with accuracy. Namely we define a function get_loss that takes as inputs the parameters and returns the loss function, and we vectorize it, and we define a meshgrid to plot a heatmap of the loss in terms of b and w. The big difference with accuracy is that the loss function varies continuously. And if we again plot how the parameters values of the keras model evolves and that at each step, the loss of the model was improved. And here we see what actually happens when we run the command 'fit the model to the data'. Namely with a step by step procedure, keras improves the loss.

One last remark here that first I stressed that accuracy was a natural quantity to evalute our model. But I'm telling you that actually the algorithm only focuses on minimizing the loss. What happens is that in general we we decrease the loss the accuracy should improve.

As a conclusion for this part, I'm gonna try to sum the idea of gradient descent: Our Goal is to minimize the loss function For that keras uses some steps called epochs, and at each epoch, keras changes with a small modification the parameters in such a way that the loss decreases Now if you're curious you might be unsatisfied and say how really keras finds this direction to move the parameters. For that one uses mathematical method called differential calculus where you need to calculate a gradient (whence the name gradient descent), and in the context of neural networks, the way you in practice calculate the gradient is called the backpropagation algorithm But I won't enter these details.

3 Neural networks

3.1 Logistic regression again

I think this was enough theory, so let's comme back to our example. What we're gonna do in this last part is modify a little bit the question about iris species classification, and see that logistic regression has some limitation, but that these limitations can be solved if one uses neural networks with more layers. So we'll change the species and focus now on versicolor. First, we create a new column called 'isVersicolor' which takes the value 1 if the iris sample is of the species versicolor and 0 otherwise.

As we did previously, we plot a histogram of the petal width with green bins for versicolor and blue bins for not versicilor. And now let us now consider the problem of predicting if an iris sample is of the species versicolor knowing its petal width.

Since Logistic regression worked so well so far, let's keep using it. As we did before we instantiate a sequential model with Keras to compute a function which mapps x to w *x + b then we add the sigmoid layer. We compile our keras model, and we fit the model with our data, namely petal width as input and the new column versicolor as output. But here we see that the accuracy get stuck at 0.66 and the loss also seems to be around 0.63 and can't go lower. It is clearly not great compared to accuracy we got for the previous problem (remember that to classify virginica, we got an accuracy of 0.96.) So let's investigate this more and plot a histogram with the probability predictions of the model together. So here the dotted black line is the probability threshold of 0.5 and what we see is that all the red curve which is the curve of the output probabilities from the model, is under 0.5. This means that our keras model predicts that all the samples are not versicolor. That is very insatisfactory. Let's try to think about this for a moment. The curve of the sigmoid starts at 0 then goes up to 1. In particular it is a monotonic function which means that it does not go up and down. And that is clearly a big problem for the versicolor classification according to petal width because clearly, the best solution for the problem is to predict False, so a probability close to 0 for low values, then True, so around 1 for values between 1 and 2, and then again predict false, so 0 for values greater then 2. In conclusion, we want that the model predicts probablities like the curve on the left, but not like on the right. And the problem is that logistic regression can only output probabilities like on the right. So we need to change something. And for that we are gonna add some layer to our neural network.

3.2 Neural Networks with hidden layers

First let's remember what how the sequential graph for logistic regression looks like It was represented by nodes where nodes represented how far you are in the sequence of computation. For logistic regression the computation is decomposed in a sequence involving 3 nodes" in the first node there is the input x in the second node w times x + b which (which is an intermediate value that we don't care ultimately but which is necessary to make the full computation) and finally in the 3rd node we apply the sigmoid function to wx+b.

The idea of neural networks is simply to add more nodes to the computation. Let's see an example. Compared to the sequential graph of logistic regression, we have added 2 vertical layers. I've written inside the nodes the transformations. What happens is that the computation is splitted in 3 parallel paths and in each of the path and each of those paths do exactly the same thing as logistic regression. Namely x is transformed to x times w + b w except that all the parmeters w and b are different for each nodes. Hence in the upper node we get w_3x + b_3, and so on. Then on each of the nodes we apply the sigmoid function as within logistic regression. I call y_i all these intermediates values. And then we add a new layer with just one node which takes the information from the 3 previous nodes by taking a weighted sum y1 times W1 + y2 times W2 + y3 times W3 + B. And finally we apply the sigmoid function to this last node.

Here This model is what is called a neural network with one hidden layer. Where the union of the layer with wixi+b_i and the sigmoid forms together the hidden layer. Compared to the logistic regression above, it depends on many more parameters. I total for each node in the hidden layer, we have 2 parameters and since there are 3 nodes we get 6 parameters, and for the node here we have 4 parameters Capital w1, capiatl w2, capital 23 and BB, so a total of 10 parameters. Now it is very easy to implemente this neural network in keras following the description of the neural network as a sequence of components (or layers). We start sying we want an object of the class sequential, then we ad the first layer, which is a Dense layer as for logistic regression, but this time we specify 3 which means that there will be 3 nodes in this layer. Now maybe the terminology Dense might be clearer: dense means that you connect all your nodes with all the nodes from the previous layer. Then we add the layer with the sigmoid functions. Then we add again a desnse layer of the form Dense(1), which means: add one node, a create

a dense connection with the all the nodes from the previous layer but add the parameters w1 w2 w3 b to create a combination from them. Finally we add one last sigmoid function.

Now that we have implemented our neural network with Keras, we fit the model to our data, and we will plot the probabiilty output at very stages of the epoch process. For this we plot 6 plots which plot the probability output after 300 epochs, then 600 ... untill 1800 epochs. And I Think that what we see is quite remarkable, in the sense that we see the curve changing epochs after epochs to really adapt to the distribution we have.

The loss function is very small and the accuracy funtion close to 1.

And now we can plot together the porbability that the model outputs and the histogram of the distribution and really see that the model does very well.

Conclusion

Before finishing, I want to recap some of the things we've been discussing:

- Neural networks are models which depend on parameters
- Logistic regression is the simplest example of a neural networks. It works well for some problems, but for some other problems it does not work well
- This probelm can sometimes been solved by adding some layers to the neural network (but it becomes slower)
- Behind the hood, keras try to fit the paramters of the neural network with the data we give to it, and for this it tries to minimize the loss function
- Keras is an easy interface to use neural networks