

How to build a neural network playground from scratch

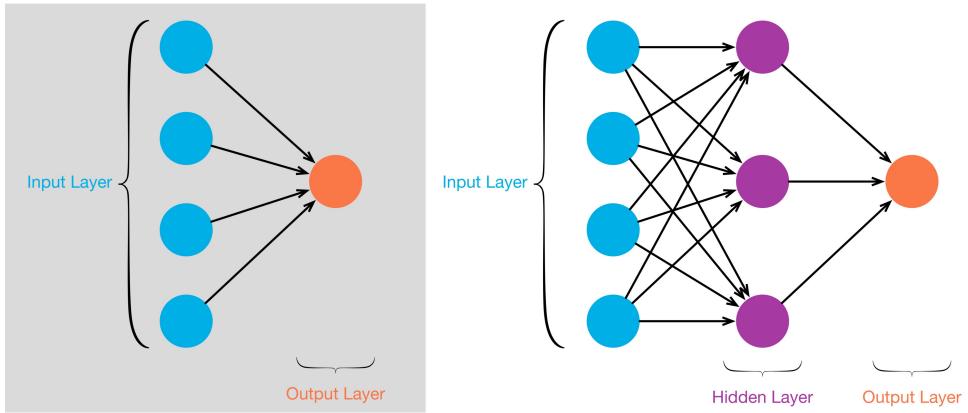


1 Intro

Neural networks became increasingly popular as shown by the research publication topics of the recent years. The main advantage of those models is that they can learn complex structures and are highly customizable. However, one significant drawback is their poor explainability power. Neural networks are often categorized in the so-called "black box models" that are hard to understand. Fortunately, there are some very good tutorials to understand how a neural network actually works. The tutorials of Andrew Ng is an example of some very good materials available online.

I personally find that a 1-layer network is relatively easy to understand, however it gets tricky when increasing the number of layers. This article is an attempt to understand in details the 2-layer neural network through the building of a small playground. We will even publish the playground online! To enjoy fully this article, the reader must be familiar with the working of a 1-layer neural network.

The difference between a 1-layer network and a 2-layer network relies in the following picture:



The 2-layer neural network has a hidden layer composed of hidden units (the neurons). Note that the input layer is usually ignored when counting the layers.

To understand deeply the 1-layer I strongly recommend Andrew Ng's tutorial for more details or this article for a wrap up.

Let's now go into deeper details on the working on a 2-layer network.

2 2-layer NN

There are already an elephantic number of articles and resources on the maths behind neural networks, so I won't go into much details but rather focus on the points that are, in my view, the most important.

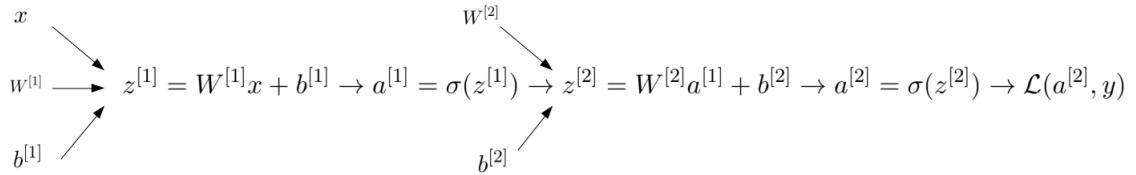
For the sake of simplicity, we will use a binary classification to build the playground. Thus, the labels belong to $\{0, 1\}$. The cost function is:

$$\mathcal{L}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

Where y is the true label and \hat{y} its estimation (probability).

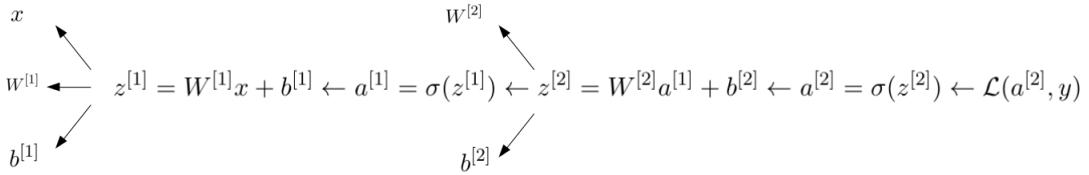
We notice that the loss function decreases when $y \neq \hat{y}$, which is expected as we want to penalize wrong classification.

The forward propagation allows to compute the loss functions based on the weights:



As we can see here, we use two sets of weights and biases: $W^{[1]}, b^{[1]}, W^{[2]}$ and $b^{[2]}$.

Then, the backward propagation allows to find how to update the weights in order to minimize the loss:



Starting from the loss' derivative and thanks to the chain rule, we end up with expressions that tell us how to update the weights to minimize the loss.

For the full details on the derivative computation, I explain everything on my website (hopefully I didn't do too many mistakes).

3 Building the app

The app will allow the user to provide two types of input:

- the shape of the data: blobs, circles or moons
- the number of hidden units in the neural network

Of course, this is just an example of what types of interactions you can have between the user and the neural network. But as explained in the conclusion of the article, you can add much more parameters!

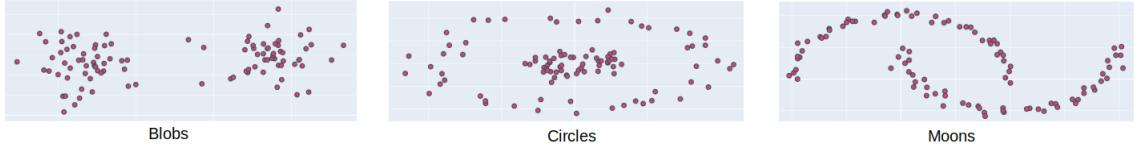
The chosen graphical library for the playground is plotly-dash. I am personally a big fan of this framework as the documentation is quite complete and the community very active.

When developing an app, it is a good practice to separate the modelling part from the graphical part. We will also create a specific module to generate the data. In the end there are 3 python scripts in this app:

- app.py: this script creates the server and contains everything about the graphical part (buttons, titles etc.)
- model.py: here we implement the neural network itself!
- utils.py: in this module we simulate data based on the user's preferences

3.1 utils.py

We will allow the user to choose between three types of data shape: blobs, circles and moons.



3.2 model.py

In this module we build the neural network logic. As mentioned earlier, we want to build it from scratch so that we understand everything what is happening.

The tricky part here is to be consistent with the dimensions, especially during the derivative computation:

Listing 1: Backward propagation

```
dz2 = a2-y_train
dw2 = (1/n_samples)*np.dot(dz2, a1.T)
db2 = (1/n_samples)*np.sum(dz2)
dz1 = np.multiply(np.dot(self.weights_2.T, dz2), 1-np.power(a1, 2))
dw1 = (1/n_samples)*np.dot(dz1, X_train.T)
db1 = (1/n_samples)*np.sum(dz1, axis = 1, keepdims = True)
```

3.3 app.py

This module creates the server and handle all graphical components. We specify the location of texts, buttons and the figure. One remark is that we use the same function (also called 'callback functions') for the two buttons. Indeed, it seems plotly-dash doesn't allow to handle different buttons with the same output (= update the graph).

4 Deployment

The deployment is done via Heroku. The clear steps for deployment are explained here. As mentioned in the link, "Heroku is one of the easiest platforms for deploying and managing public Flask applications.".

To sum up, the deployment consists in hosting the code in a Github repo, then deploying through Heroku with this command:

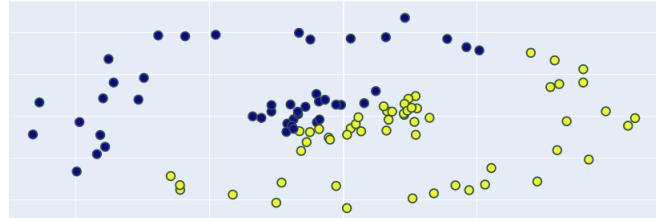
Listing 2: Deploy the app

```
$ git push heroku master
```

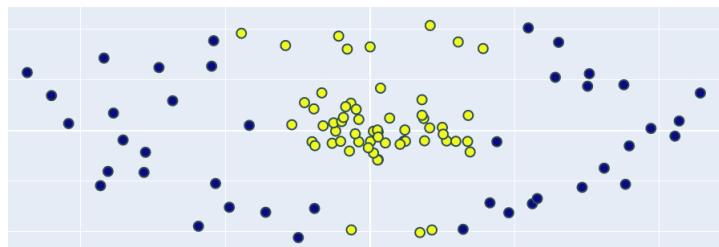
5 Key insights

When playing with the app we can already draw some interesting insights:

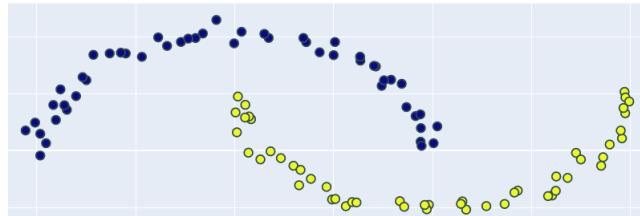
- 1 hidden unit gives a linear separation through one line only. This is especially visible when selecting circles.



- 2 hidden unit gives a linear separation through two lines and thus allow a better separation.



- 3 hidden units perfectly separates the data when the shapes are not convex (moons).



This illustrates the power of neural networks. We can think of how good they can be when identifying human faces or finger prints!

6 Extensions

The extension possibilities are endless! You can think about adding checkboxes/buttons so that the user can choose:

- the noise (standard deviation) associated with the data generation
- the number of data points to be generated
- the learning rate to be used during the parameter update
- the number of layers

For the last point however, I believe it would require significantly more work, especially if you don't use any neural network library (like tensorflow or pytorch) as we do here.

7 Conclusion

For the final version of the playground, check out this link. All the code is available on my github repo.

Have fun!

PS: This is my first article on Medium, I'd be happy to get your feedback with a clap and/or a comment.