Neural Network Study

Manuel Naviglio

In this brief note we discuss the output distribution of a Multilayer Perceptron (MLP) Neural Network given different inputs. An MLP is a type of feedforward neural network that consists of multiple layers of artificial neurons. Each neuron in an MLP takes one or more inputs and produces a single output, which is then fed forward to the next layer of neurons. The input layer receives the raw input data, and the output layer produces the final output of the network. The neurons in the hidden layers use activation functions to transform the weighted sum of their inputs into a non linear output. The weights of the connections between neurons in each layer are learned during training using an optimization algorithm like back propagation.
Let us suppose that we have an N-dimensional input

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_N \end{pmatrix}, \tag{1}$$

and that the first hidden layer is composed by K neutrons. The first layer takes the vector input and multiplies it by a weights matrix summing a bias vector. The output of the first hidden layer is

$$y_1^i = f_1(W_1^{ij} x^j + b_1^i), \qquad i = 1, ..., K, \tag{2}$$

where W_1 is a $N \times K$ weights matrix (in other words it has lines equal to the dimension of the input and columns equal to the number of neurons) while b_1 is a N-dimensional bias vector. The function f_1 is called activation function and it has the role of introducing non linearity between the different layers. neuron. Without a non linear activation function, the network would be limited to linear transformations of the input data, which would make it unable to effectively model complex relationships between inputs and outputs. By applying a nonlinear activation function, the output of each neuron is transformed in a way that is dependent on the input to that neuron. This allows the network to model non linear relationships between the input and output. Different activation functions have different properties that make them more or less suitable for different tasks. We discuss different choices later.
If we consider L layers we can represent an MLP as follows:

$$y_L = W_L f_{L-1}(W_{L-1} ... f_2(W_2 f_1(W_1 x + b_1) + b_2) + ... + b_{L-1}) + b_L. \tag{3}$$

The weights and biases have to be initialized. Then, we can train the NN using data and they will assume the values that minimize the loss function that we chose. Let us now neglect the training of the neural network and let us consider only the initialization configuration of the output. In particular, let us suppose that we initialize the weights matrix and the bias matrix with values taken by a certain probabilistic distribution function, i.e. a Gaussian distribution. If we consider a unidimensional input x, then the output of the first hidden layer is simply

$$y_1^j = f_1(W_1^j x + b_1^j) = f_1\left((x) \begin{pmatrix} w_1 & w_1^2 & ... & w_1^K \end{pmatrix} + \begin{pmatrix} b_1^1 \\ b_1^2 \\ ... \\ b_1^K \end{pmatrix} \right) = \begin{pmatrix} f_1(w_1^1 x + b_1^1) \\ f_1(w_1^2 x + b_1^2) \\ ... \\ f_1(w_1^K x + b_1^K) \end{pmatrix}. \tag{4}$$

Thus, the output of the hidden layer is a vector having dimensions equal to the number of neurons of the layer.

If we suppose that the NN has only one hidden layer and an unidimensional output, the output is simply represented by a final layer having a single neuron. It has the following form:

$$y_2 = f_2\left(y_1^i W_2^i + b_2\right) = \begin{pmatrix} y_1^1 & y_1^2 & ... & y_1^K \end{pmatrix} \begin{pmatrix} w_2^1 \\ w_2^2 \\ ... \\ w_2^K \end{pmatrix} + b_2 = \sum_{k=1}^{K} y_1^k w_2^k + b_2 \\ = \sum_{k=1}^{K} f_1(w_1^k x + b_1^k) w_2^k + b_2. \tag{5}$$

Every number w and b can be initialized taking the values from a Gaussian distribution. If do this T times we obtain T different values of the output y_2. The question that we want to answer is how the distribution of the T values of y_2 looks like as a function of different choices of the parameters of the neural network. In particular the main question is when the final distribution is Gaussian.

1 Gaussian initialization

The parameters of the NN that we can vary are the mean μ and the variance σ of the input distribution, the number of layers L (deepness of the NN), the number of neurons K for each layer (size of the NN) and the activation function f. For simplicity we only consider an unidimensional input x.
In this Section we take the values that fill the matrices and the vectors of the weights and biases from a Gaussian distribution

$$f_L(x) = \frac{1}{\sigma_L(K)\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma_L(K)^2}\right) \tag{6}$$

where the variance of the Lth layer, namely $\sigma_L(K)$, is a function of the number of neurons contained in the previous layer K and of the value of the variance of the previous layer σ_{L-1}. In this work $\sigma_L = \sigma_{L-1}/K$. Differently, the mean μ remains the same for each layer. Thus, for example the T values of w_1^1 will be gaussian distributed. In this work $T = 10000$.

To test if the final distribution is normal we use the "normaltest" function of the "scipy.stats" module in Python. The normaltest function tests the null hypothesis that a sample of data comes from a normal distribution. It does this by computing the skewness and kurtosis of the sample, and then performing a statistical test to determine whether these values are consistent with those expected for a normal distribution. The function makes following the steps:

- Compute the skewness and kurtosis of the sample data. Skewness measures the symmetry of the distribution, and kurtosis measures the "peakedness" or "flatness" of the distribution. For a normal distribution, the skewness should be close to 0, and the kurtosis should be close to 3;
- Compute the test statistic, which is a combination of the skewness and kurtosis values. The test statistic is computed as $statistic = skewness^2 + kurtosis^2$. This test statistic is expected to follow a chi-squared distribution with 2 degrees of freedom under the null hypothesis of normality;
- Compute the p-value associated with the test statistic. This p-value represents the probability of observing a test statistic as extreme or more extreme than the one computed from the sample, assuming the null hypothesis of normality is true;
- Compare the p-value to a significance level (e.g., 0.05). If the p-value is less than the significance level, we reject the null hypothesis of normality and conclude that the data is not likely to be normally distributed. If the p-value is greater than the significance level, we fail to reject the null hypothesis and conclude that the data is likely to be normally distributed.

We consider a significance level of 0.05.

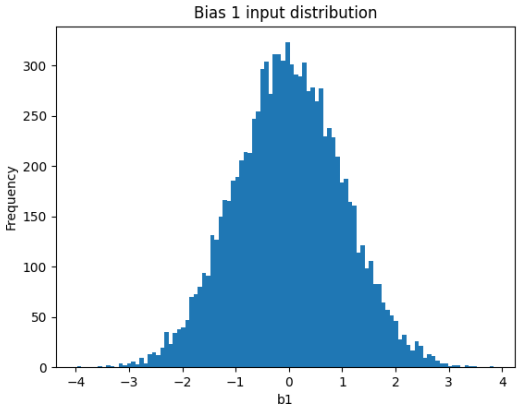

Figure 1: *Input distribution of one entry of the bias in the case* $\mu = 0, \sigma = 1$.

1.1 One Hidden Layer Neural Network

Let us start by taking the simple case $\mu = 0, \sigma = 1$ with a ReLU activation function and a single hidden layer. The output distribution as a function of the size of the layer can be observed in Fig. **??**. The output starts to become nearly gaussian already around $L = 80$.
In the case of a Sigmoid activation function the output distribution is already gaussian with 2 neurons as it can be observed in Fig. **??**. Choosing a Hyperbolic tangent or a Softmax activation function has the same effect. Differently, as it can be expected, a Leaky ReLU or an ELU activation function give the same effect as in the ReLU case.

If we try to keep fixed the σ and increase the mean value, for example $\mu = 60$, the gaussianity seems to be reached faster. See Fig.s **??** and **??**.

Let us now fix both the mean $\mu = 10$ and the number of neurons for the hidden layer $L = 5$. From Figure **??** it can be clearly observed that as we increase σ the final distribution becomes less gaussian. In any case, also for a fixed large value $\sigma = 10$ the gaussianity is reached by increasing the number of neurons, see Fig. **??**.
This study probably tells us that as the relative error increases the number of neurons necessary to obtain a gaussian output increases.

1.2 Multiple Hidden Layers Neural Network

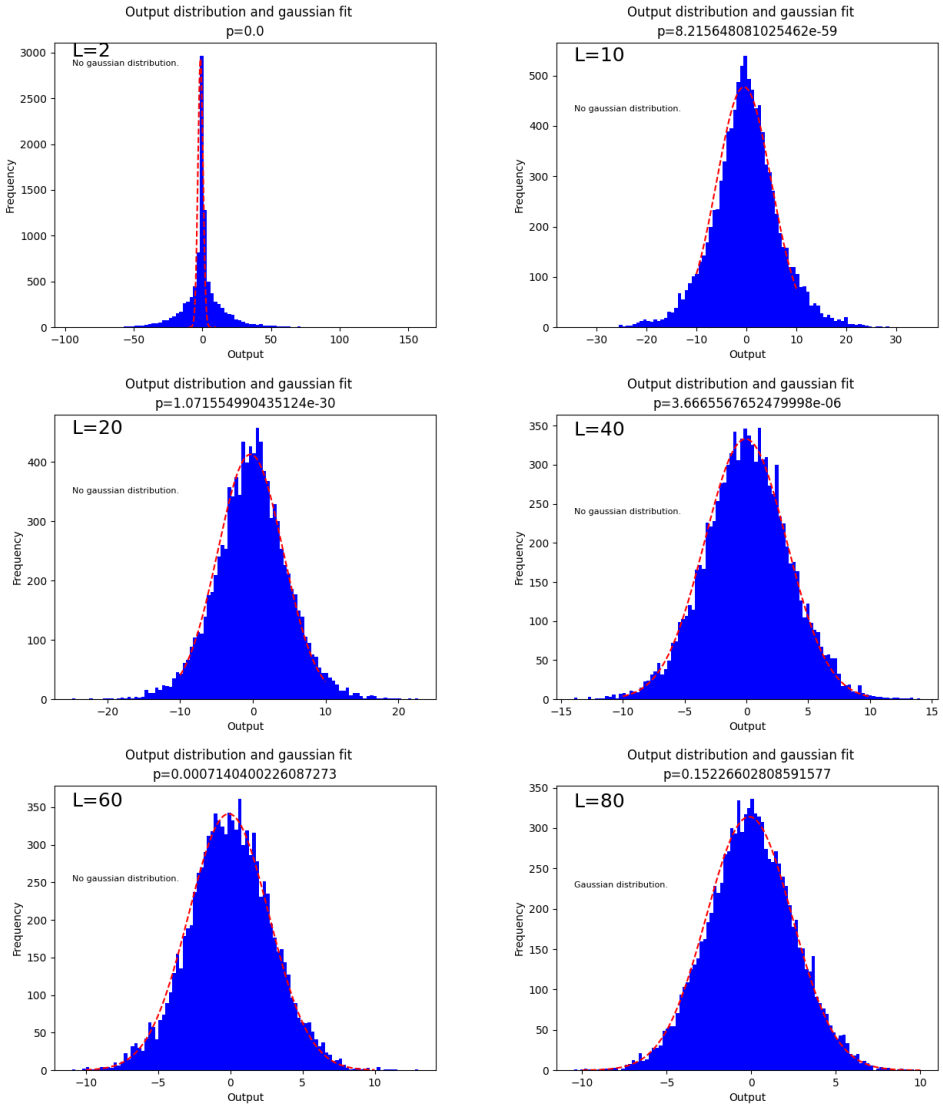

Figure 2: *Output distribution in the case* $\mu = 0, \sigma = 1$ *with one hidden layer and a ReLU activation function as a function of* L*.*

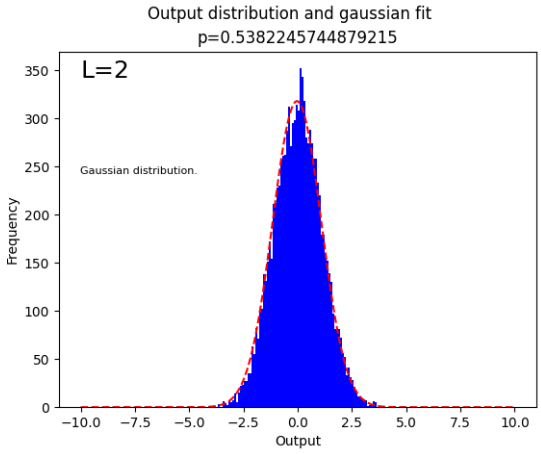

Figure 3: *Output distribution in the case* $\mu = 0, \sigma = 1$ *with one hidden layer and a Sigmoid activation function as a function of* L*.*

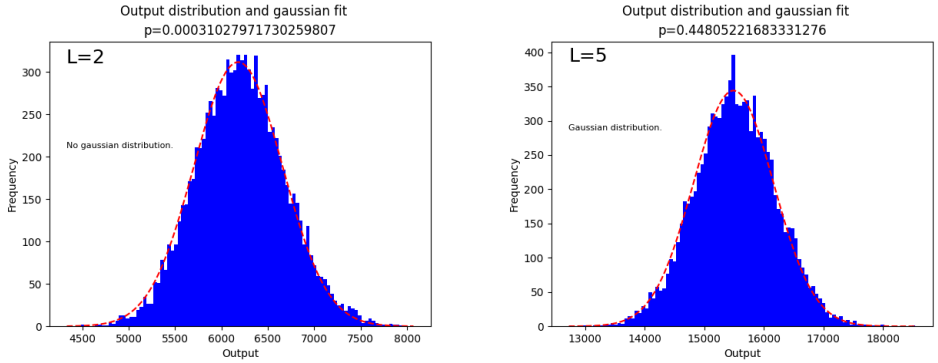

Figure 4: *Output distribution in the case* $\mu = 10, \sigma = 1$ *with one hidden layer and a ReLU activation function as a function of* L*.*

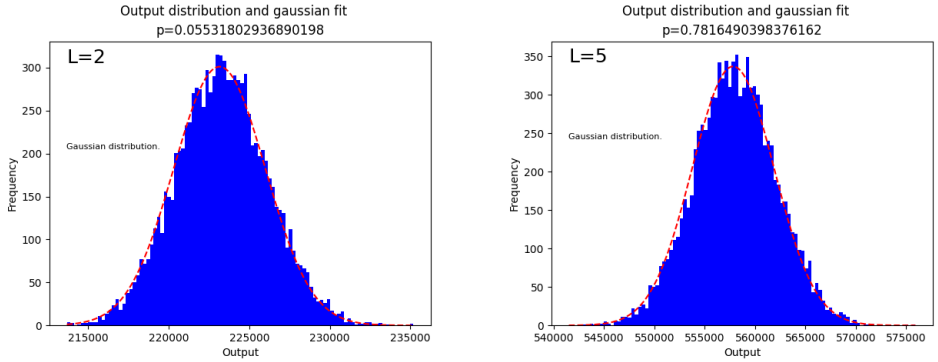

Figure 5: *Output distribution in the case* $\mu = 60, \sigma = 1$ *with one hidden layer and a ReLU activation function as a function of* L*.*

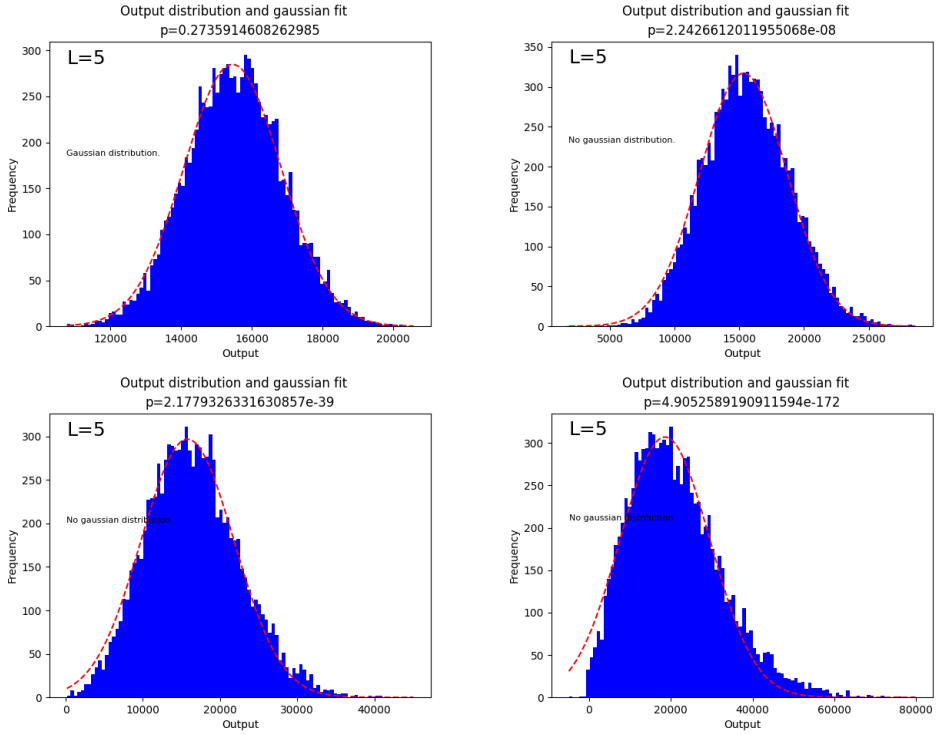

Figure 6: *Output distribution in the case* $\mu = 60, L = 5$ *with one hidden layer and a ReLU activation function as a function of* σ*. Respectively we are considering the values* $\sigma = \{2, 5, 10, 20\}$*.*

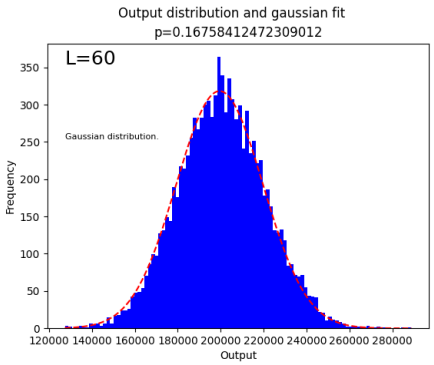

Figure 7: *Output distribution in the case* $\mu = 10, \sigma = 10$ *with one hidden layer and a ReLU activation function with* $L = 60$.