Feature Perturbation With Neural Networks.

Bhavesh Khamesra

February 2021

1 Introduction

With the emergence of deep learning, neural networks have found applications in almost every field ranging from financial analytics to computer vision and speech recognition. However, a common problem cited with this machine learning model is interpretability - due to multiple non-linear transformations, it is difficult to understand the final result and what characteristics of the initial data led to it. Feature perturbation can help in analysing the effects of individual features on the final results. This is a simple and extremely efficient method which can allow deep learning researchers to comprehend the black box neural networks by directly relating the changes in input features with the final output. Such a method can have wide applications in various domains ranging from credit monitoring, banking solutions, sales forecasting, risk management etc. Let's consider a few examples.

Loan Applications - Suppose a customer applies for a loan from a bank. The bank collects a range of information from your application and other services/vendors such as your financial and liquid assets, credit scores, credit history, number of defaulted payments etc and analyzes the data using machine learning models which rejects your loan application. The question of interest is what specific features in customer's application led to the rejection and what improvements are required to get the loan approved?

Stock Prediction - The stock price of a company relies on several factors ranging from company portfolio, annual performance to daily trading exchanges. As a result the prices of stock varies continuously based on the changes in these factors. In order to make forecast, one needs to understand the impact of each feature on the final output. An alternate problem of higher interest for the traders and companies would be to understand which specific feature variations resulted in changes of stock price and if this can be quantified?

2 Problem Setup -

Consider a feed forward neural network with N hidden layers and M inputs. The neural network is already trained with known weights and biases. To simplify the problem, we consider a binary classifier and assume the activation function for all the hidden layers is ReLU while for output layer is sigmoid. Consider a case where the output of the classifier is 0. We want to determine the minimum required changes in the input features which can change the output to 1.

Let's define some notation which will be used in deriving the solution -

Notation:

- Let l represent the l^{th} layer index and n_l represent the total number of nodes in l^{th} layer.
- $w_{jk}^{[l]}$ is the weight assigned to k^{th} input from previous layer, $l-1$, to compute output of j^{th} node of layer l.
- $b_j^{[l]}$ is the bias term corresponding to jth node of layer l.
- $z_j^{[l]}$ represents the linear part of the node before applying the activation function. j corresponds to the node index while l corresponds to the layer index.
- $Z^{[l]}$ represents the matrix of dimensions $(n_l, 1)$ with elements $z_j^{[l]}$, $j = 1....n_l$.
- $g^{[l]}(.)$ represents the activation function of layer l.
- $a_j^{[l]}$ represents the output of node j of layer l.
- $A^{[l]}$ is the matrix of dimensions $(n_l, 1)$ with $a_j^{[l]}$ as the elements, $j = 1....n_l$

2.1 Setup:

We now feed our neural network with test data X with elements $x_k, k = 1.....M$. Let's look at the forward propogation equations. For I layer -

$$z_j^{[1]} = \sum_{k=0}^{n_0} w_{jk}^{[1]} a_k^{[0]} + b_j^{[1]} \tag{1}$$

$$a_j^{[1]} = g^{[1]}(z_j^{[1]}) \tag{2}$$

Here, $n_0 = M$ and $a_k^{[0]} = x_k$. Generalizing this to l^{th} layer -

$$z_j^{[l]} = \sum_{k=0}^{n_{l-1}} w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[1]} \tag{3}$$

$$a_j^{[l]} = g^{[l]}(z_l^{[l]}) \tag{4}$$

In matrix form, we can write these equations as

$$A^{[l]} = g^{[l]}(Z^{[l]}) \tag{5}$$

where $A^{[l]}$ is the $(n_l, 1)$ output of l^{th} layer and $Z^{[l]}$ is a column vector of same dimensions with elements $z_j^{[l]}$ where $j = 1....n_l$. In matrix form, the equation for $Z^{[l]}$ takes the form -

$$Z^{[l]} = W^{[l]} A^{[l-1]} + B^{[l]}$$

where $W^{[l]}$ corresponds to weight matrix with dimensions (n_l, n_{l-1}) and $B^{[l]}$ is the bias vector with dimensions $(n_l, 1)$. For hidden layers, $g^{[l]}(.)$ corresponds to ReLU function while for last layer unit, $g^{[N]}() = \sigma^{[N]}()$ is the sigmoid function. If T represents the threshold, then

$$\hat{y} = \begin{cases} 1, & \text{if } A^{[N]} \geq T \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

2.2 Single Feature Perturbation -

Now consider the case where the input vector is classified as 0. Let's assume the classification occurs primarily due to the i^{th} feature value x_i (this will be more clear when we consider an example) and we are interested in determining the required change in x^i which would alter the result. Let's change x_i to $x_i + \delta x_i$ while keeping other features constant, and compute the corresponding changes in the final output using forward propagation. We will represent the modified quantities by tilde. For Layer 1, we get -

$$\tilde{z}_j^{[1]} = \left(\sum_{k=1}^{n_0} w_{jk}^{[1]} a_k^{[0]} + b_j^{[1]} \right) + w_{ji}^{[1]} \delta x_i \tag{7}$$

$$= z_j^{[1]} + w_{ji}^{[1]} \, \delta x_i \tag{8}$$

$$= z_j^{[1]} + \delta z_j^{[1]} \tag{9}$$

where $\delta z_j^{[1]} = w_{ji}^{[1]} \, \delta x_i$. Applying the activation function and using Taylor expansion, we get

$$\tilde{a}_j^{[1]} = g^{[1]} \left(\tilde{z}_j^{[1]} \right) \tag{10}$$

$$= g^{[1]} \left(z_j^{[1]} + \delta z_j^{[1]} \right) \tag{11}$$

$$= g^{[1]}(z_j^{[1]}) + \sum_{k=1}^{n_1} \frac{\partial g_j^{[1]}}{\partial z_k^{[1]}} \, \delta z_k^{[1]} \tag{12}$$

$$= a_j^{[1]} + \delta a_j^{[1]} \tag{13}$$

Here, in step 3, $\frac{\partial g_j^{[1]}}{\partial z_k^{[1]}} = \frac{\partial g^{[1]}(z_j^{[1]})}{\partial z_k^{[1]}}$ and in step 5, we define $\delta a_j^{[1]} = \sum_{k=1}^{n_1} \frac{\partial g_j^{[1]}}{\partial z_k^{[1]}} \, \delta z_k^{[1]}$. Thus, the perturbation in feature x^i would affect all the nodes of first layer to different degrees determined by the weight and the activation function.
Let's generalize this expression for l^{th} layer.

$$\tilde{z}_j^{[l]} = \sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} \tilde{a}_k^{[l-1]} + b_j^{[l]} \tag{14}$$

$$= \sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} \left(a_k^{[l-1]} + \delta a_k^{[l-1]} \right) + b_j^{[l]} \tag{15}$$

$$= z_j^{[l]} + \sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} \delta a_k^{[l-1]} \tag{16}$$

$$= z_j^{[l]} + \delta z_j^{[l]} \tag{17}$$

$$\tag{18}$$

The output of $j^{th} node$ of l^{th} layer then becomes

$$\tilde{a}_j^{[l]} = g^{[l]}(z_j^{[l]} + \delta z_j^{[l]}) \tag{19}$$

$$= a_j^{[l]} + \sum_{k=1}^{n_l} \frac{\partial g_j^{[l]}}{\partial z_k^{[l]}} \, \delta z_k^{[l]} \tag{20}$$

$$= a_j^{[l]} + \delta a_j^{[l]} \tag{21}$$

where in the second step, we again perform Taylor expansion. From above equations, we can obtain the following recursive relation for $\delta z_j^{[l]}$ -

$$\delta z_j^{[l]} = \sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} \sum_{m=1}^{n_{l-1}} \frac{\partial g_k^{[l-1]}}{\partial z_m^{[l-1]}} \, \delta z_m^{[l-1]} \tag{22}$$

In matrix form, we can write these equations as

$$\delta Z^{[l]} = W^{[l]} \, \nabla_{Z^{[l-1]}} G^{[l-1]} \; \delta Z^{[l-1]} \tag{23}$$

where $\nabla_{Z^{[l]}} G^{[l]}$ is a (n_l, n_l) dimensional matrix whose elements are $\partial g_k^{[l]} / \partial z_m^{[l]}$. Here, we note that for ReLU functions, the second or higher order derivatives are zero and hence, we can stop the Taylor expansion at the first derivatives. However, sigmoid is a infinitely differentiable function (C^∞) and hence, using same strategy would ignore higher order derivatives which can result in significant errors. Hence, for the last layer, we apply slightly different approach. Using expression 18 for last layer with single node ($j = 1$), we have

$$\tilde{z}_1^{[N]} = z_1^{[N]} + \delta z_1^{[N]} \tag{24}$$

$$\tilde{a}_1^{[N]} = \sigma\left(\tilde{z}_1^{[N]}\right) \tag{25}$$

$$= \sigma\left(z_1^{[N]} + \delta z_1^{[N]}\right) \tag{26}$$

Now, under our assumption, output of $x_i + \delta x_i$ will change to $\hat{y} = 1$ if $\tilde{a}_1^N \geq T$. Substituting $\tilde{a}_1^{[N]}$ from above expression, we have

$$T \leq \tilde{a}_1^{[N]} = \sigma(\tilde{z}_1^N) \tag{27}$$

$$\leq \frac{1}{1 + e^{-\tilde{z}_1^{[N]}}} \tag{28}$$

$$\tag{29}$$

After some algebra and using log function, we get

$$\tilde{z}_1^{[N]} \geq -log\left(\frac{1-T}{T}\right) \tag{30}$$

$$\delta z_1^{[N]} \geq -log\left(\frac{1-T}{T}\right) - z_1^{[N]} \tag{31}$$

We can now use the recursive relations obtained in equation 22 to find the relation between $\delta z_1^{[N]}$ and δx_i. Since final layer has single node, $\delta z_1^{[N]}$ will be same as $\delta Z^{[N]}$. Hence, we can use the matrix

form of this equation as given in equation 23

$$\delta Z^{[N]} = W^{[N]} \nabla_{Z^{[N-1]}} G^{[N-1]} \delta Z^{[N-1]} \tag{32}$$

$$= W^{[N]} \nabla_{Z^{[N-1]}} G^{[N-1]} W^{[N-1]} \nabla_{Z^{[N-2]}} G^{[N-2]} \delta Z^{[N-2]} \tag{33}$$

$$= \left(\prod_{i=0}^{N-2} W^{[N-i]} \nabla_{Z^{[N-i-1]}} G^{[N-i-1]} \right) \delta Z^{[1]} \tag{34}$$

From equation 8, we have $\delta z_j^{[1]} = w_{ji}^{[1]} \delta x_i$ or in matrix form $\delta Z^{[1]} = V^{[1]} \delta x_i$ where $V^{[1]}$ is a column vector with elements $w_{ji}^{[1]}$, $j = 1....n_1$. Substituting in above equation, we get

$$\delta Z^{[N]} = \left(\prod_{i=0}^{N-2} W^{[N-i]} \nabla_{Z^{[N-i-1]}} G^{[N-i-1]} \right) V^{[1]} \delta x_i \tag{35}$$

Substituting this relation on LHS of equation 31, we get the following result for δx_i

$$\delta x_i \geq -\left(log\left(\frac{1-T}{T}\right) + z_1^{[N]} \right) \left[\left(\prod_{i=0}^{N-2} W^{[N-i]} \nabla_{Z^{[N-i-1]}} G^{[N-i-1]} \right) V^{[1]} \right]^{-1} \tag{36}$$

The above method assumes the ideal case where neural network is able to perfectly determine the underlying decision boundaries. In such a case, δx_i would essentially measure the projected distance of x_i from the nearest point on the decision boundary. However, in real world applications, we are limited by the the accuracy of neural network, available number of data points and noise in datasets. As a result, the sample distribution may not correctly reflect the true probability distribution and/or the neural network may not represent the true decision boundary. This may lead to incorrect predictions of δx_i.

Further, the current method relies on Taylor expansion the accuracy of which depends on the number of terms included in the series. For ReLU, we got around this issue by using the property that higher order derivatives of ReLU are zero and hence, the solution is exact. However, one of the problems with Relu is the point of discontinuity at 0 and hence, if the change input parameter $|\delta z_i^{[l]}|$ is very large s.t. $sgn(z_i^{[l]}) \neq sgn(z_i^{[l]} + \delta z_i^{[l]})$, than this method will fail as one would encounter the discontinuity at $z_i^{[l]} = 0$. To tackle this issue, at present we apply this approach iteratively.

3 Multi-Feature Perturbation -

If we perturb multiple features s.t. more than one element of $\delta X = [\delta x_i]_{n_1 \times 1}$ are non-zero, then we require additional system of equations to determine δX correctly. For this, we will minimize the distance of the features from its decision boundary. If we consider N dimensional feature space, where decision boundary segregates the clusters of point with 0 or 1 class output, then in order to change the output class, one needs to cross the decision boundary. To achieve this in optimal way, we can find the closest point on the decision boundary w.r.t the given point by minimizing this distance. Then, the required change in each feature δX_i will just be the projection of this distance on corresponding feature axis. Let's now see how this can be done -

Let's assume δX represents the vector containing smallest change required in each element to change the final output. By construction, the distance of X from the decision boundary will be $\sqrt{(\delta X)^T (\delta X)}$. However, δX is also constrained by above equation. Thus we have to minimize a

function with given constraint which can be achieved with Lagrange Multiplier method. We start with following function -

$$F(\delta X, \lambda) := \delta X^T \delta X + \lambda \left[P^{L1} \delta X - \left(\log\left(\frac{1-T}{T}\right) - Z^L \right) \right]$$

And following Lagrange's method, we set the derivatives $dF/d\lambda = 0$, $dF/dX_i = 0$. Solving these equations yield -

$$\delta X = \frac{P^{L1}}{\sum_{i=1}^{N} (P_i^{L1})^2} \left(\log\left(\frac{1-T}{T}\right) - Z^L \right)$$

As a check, we see that above equation recovers the expected result for single feature perturbation.