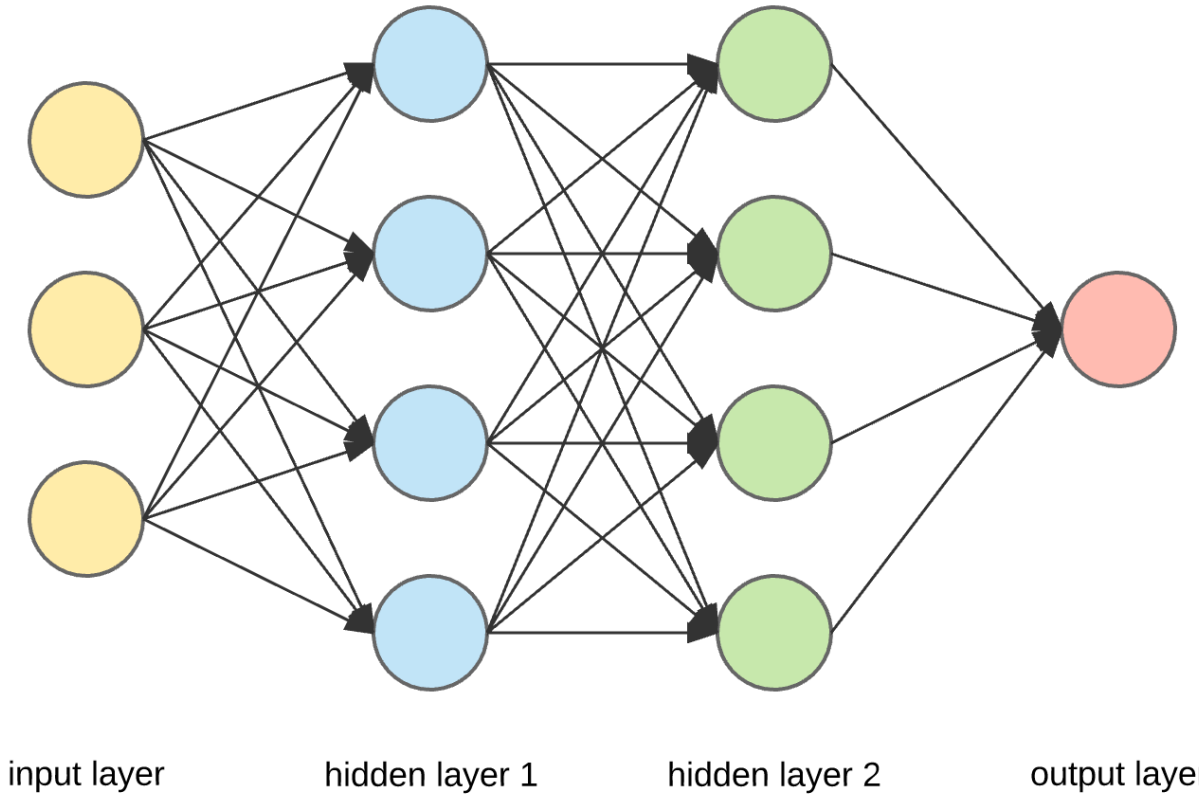
Neural Networks

Aaryan
CO21BTECH11001

A Neural Network is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.
A Neural Network consists of neurons, which, except the input neurons, accept a input and fire whenever they learn specific patterns from the data, and we model the fire rate using an activation function.
They consist of an input layer, hidden layer(s) and an output layers, a layer is a combination of multiple neurons.



The network consists of connections, each connection transferring the output of a neuron to the input of another neuron.

The output of a neuron y, given its input x can be represented as following, if we assume a linear relationship between the input and the output.

$$y = w^T x + b$$

where w, b represents the weight and bias respectively.

Let us use sigmoid function as activation function.

Forward Propagation:

Counting scheme: Input layer is considered as 0^{th} layer and layers beyond that are indexed as $1^{st}, 2^{nd}, \ldots$ layers.

For 1^{st} layer, $z^{[1]} = w^{[1]}x + b^{[1]}$

For $l > 1$ layer, $z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$

where $a^{[l]}$ stands for activation of layer l.

$$a^{[l]} = \sigma\left(z^{[l]}\right) \forall l$$

Optimization:

Define loss/cost function

$$J(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m} L^{(i)}$$

$$L^{(i)} = -\left[y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)\right]$$

Backward Propagation:

To optimize for $w^{[l]}\ \forall l$, we use gradient descent.

$$w^{[l]} = w^{[l]} - \alpha \frac{\partial J}{\partial w^{[l]}}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}}$$

Chain rule of gradients is used to find the gradients.

Let last layer is L^{th} layer. Then,

$$\frac{\partial J}{\partial w^{[L]}} = \frac{\partial J}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial w^{[L]}}$$

$$\frac{\partial L^{(i)}}{\partial w^{[L]}} = -\left[y^{(i)} \frac{\partial \left(\log\left(\sigma\left(w^{[3]} a^{[2]} + b^{[3]} \right) \right) \right)}{\partial w^{[L]}} + \left(1 - y^{(i)} \right) \frac{\partial \left(\log\left(1 - \sigma\left(w^{[3]} a^{[2]} + b^{[3]} \right) \right) \right)}{\partial w^{[L]}} \right]$$

$$= (a^{[L]} - y^{(i)}) a^{[L-1]^T}$$

$$\frac{\partial J}{\partial w^{[L]}} = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - a^{[L]}) a^{[L-1]^T}$$

$$\frac{\partial a^{[l]}}{\partial z^{[l]}} = a^{[l]}\left(1 - a^{[l]}\right) \forall l, \because \sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$$

$$\frac{\partial z^{[l]}}{\partial w^{[l]}} = a^{[l-1]^T} \ \forall l > 1$$

For $l = 1, \frac{\partial z^{[1]}}{\partial w^{[1]}} = x^T$

$a^{[l]}(1 - a^{[l]})$ represents element by element multiplication.

Now $\frac{\partial J}{\partial w^{[L-1]}}$ can be found by chain rule

$$\frac{\partial J}{\partial w^{[L-1]}} = \frac{\partial J}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \frac{\partial z^{[L-1]}}{\partial w^{[L-1]}}$$

where $\frac{\partial J}{\partial z^{[L]}} = \frac{\frac{\partial J}{\partial w^{[L]}}}{\frac{\partial z^{[L]}}{\partial w^{[L]}}}$ and so on.

Improving Neural Networks :

1. **Using activation functions:**
   We can use different activation functions for different dataset.
   For example, ReLU, tanh(x), sigmoid function etc.

2. **Initialization methods:**
   We can transform data into zero mean and unit variance by:

$$X := X - \mu, where\ \mu = \frac{1}{m}\sum_{i=1}^{m} X^{(i)}$$

$$X = \frac{X}{\sigma}, where\ \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}\left(X^{(i)}\right)^2$$

   Test set is also transformed by using same μ and σ.
   Other methods like Xavier Initialization also comes under this category.

3. **Optimization:**
   We can use **mini batch gradient** descent to vectorize the code, as well as maintain the accuracy.
   Let $X = \left(X^{(1)}, \dots, X^{(m)}\right), Y = \left(y^{(1)}, \dots, y^{(m)}\right)$
   We can merge $X^{(i)}s$ into groups (of say 1000) and rewrite as:

$$X = \left(X^{\{1\}}, \dots, X^{\{T\}}\right), Y = \left(y^{\{1\}}, \dots, y^{\{T\}}\right)$$

   and then apply batch gradient descent on $(X^{\{i\}}, y^{\{i\}}$

   Or we can apply **Gradient Descent + Momentum Algorithm** to speed up the optimization.

$$v := \beta v + (1-\beta)v$$
$$w := w - \alpha v$$

Questions:

1. Neural networks can be used for:
   (a) Classification
   (b) Regression
   (c) Both
   Ans. (c)

2. What is the output of the input layer?
   Ans. No computation is done here within the input layer, they just pass the information to the next layer.

3. Are the equations of forward propagation correct?
   Ans. They seem to be correct, but in terms of matrix multiplication they need to be modified. Also, sometimes two matrices are added with the help of broadcasting.

4. Why is backward propagation used in optimization?
   Ans. Since the output layer is giving the final hypothesis, therefore it is easy to compute the loss function and its gradient for last layer. After that, the same is calculated for last second layer and so on.

5. How to determine the number of layers and number of neurons in each layer of a Neural Network?
   Ans. It has to be determined by a lot of guesses. Typically, GridSearchCV from keras in python can be used to determine the best possible parameters from your guesses.
   However in a classification problem, the method generally used is to reduce the number of neurons from input to output layer.

6. What are the disadvantages of using Neural Networks?
   Ans. Trial and error is used for finding the best parameters. It is computationally expensive and has unexplained behavior, therefore they are called "Black Box".