Neural Nets

Saturday, 3 April 2021 7:22 PM

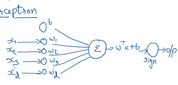
→ At a very conceptual level, our goal in supervised learning is to learn a mapping (function) between the datapoint and its label.

→ Most of the mappings we have seen are of the form $w^T x + b$, where x is a datapoint in R^d, $w \in R^d$ and $b \in R$ form the model.

→ However linear models may not work well for data we may encounter in real life situations

→ The main idea behind neural networks is to introduce "non-linearity" in the model to approximate the mapping better. The architecture of a neural network is inspired by the architecture of neurons in human brain.

Perception



Multilayer Perception / ANN

1. I/p : $x_1, x_2, \ldots x_d$ for data point x.
2. Hidden layer has two parts
   i) linear
   $a_j = \sum_{i=0}^{d} w_{ji} x_i$, $\forall j \in \{1, \ldots J\}$
   ii) Non linear Activation
   $z_j = \frac{1}{1+\exp(-a_j)}$ (Here I have used sigmoid)
3. Output
   $y = \frac{1}{1+\exp\left(-\left(\sum_{j=0}^{J} v_j z_j\right)\right)}$

We consider Loss (y, y^*) where y^* is actual label.

Design Questions for NN

1. How Deep?
2. # of neurons per layer
3. What activation?
4. What Loss/objective fn?

Activation functions :

i) Sigmoid : $\sigma(u) = \frac{1}{1+e^{-u}}$

ii) tanh

iii) ReLU : $ReLU(u) = \max(0, u)$

iv) Leaky ReLU $= \max(0.1u, u)$

Loss functions :-

For Regression

Squared error

$\frac{1}{2}(y - y^*)^2$

For Classification

Cross Entropy

$y^* \log(y) + (1-y^*)\log(1-y)$

Given data $\{x_i, y_i\}_{i=1}^{n}$

we predict $\hat{y} = f_\theta(x_i)$

and given a loss $l(\hat{y}_i, y_i) \in R^+ \cup \{0\}$

we try to get

$$\theta^* = \arg\min_\theta \sum_{i=1}^{n} l\left(f_\theta(x_i), y_i\right)$$

We do it by SGD kind of algorithm

$$\theta^{t+1} = \theta^t - \eta \nabla l\left(f_\theta(x_i), y_i\right)$$

How to do it for Neural Networks?

Ans: Back propagation

Idea: Use chain Rule of Calculus:-

If $y = g(u)$ & $u = h(x)$

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

Suppose, for example

$$f(w, x) = \frac{1}{1+e^{-(w_0 + w_1 x_1)}}$$

For given values of x_1 & w_0, w_1 you first calculate y.

Now in the backward pass you need $\frac{\partial f(w,x)}{\partial w_0}$ & $\frac{\partial f(w,x)}{\partial w_1}$

You do this by going backwards from output layer towards i/p layer & applying the chain rule to calculate gradient at each step.

Example Logistic Regression as Neural Network

$$a = \sum_{i=0}^{d} w_i x_i$$

$$y = \frac{1}{1+\exp(-a)}$$

loss $J = y^* \log y + (1-y^*)\log(1-y)$

Back prop

$$\frac{\partial J}{\partial y} = \frac{y^*}{y} + \frac{(1-y^*)}{y-1}$$

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial a}$$

Now $\frac{\partial y}{\partial a} = \frac{\exp(-a)}{(\exp(-a)+1)^2}$

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial w_i}$$

$$\frac{\partial a}{\partial w_i} = x_i$$

$$\frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial x_i}$$

$$\frac{\partial a}{\partial x_i} = w_i$$