Deep Learning with Perception

Jawwad Shamsi

September 23, 2021

1 Fundamentals of Deep Neural Networks contd

This is a continuation of the previous notes.

1.1 Activation Functions

Activation Functions add non-linearity to the o/p Following activation functions were discussed

1. Sigmod
2. Softmax
3. tanh
4. RELU
5. Leaky Relu

They are explained in detail in the previous section. Below is a summary extracted from the book

Table 2.1 A cheat sheet of the n

Activation function	Description
Sigmoid/ logistic function	Squishes all the values to a prob ity between 0 an which reduces extreme values or outliers in the data. Usually used to classify two classes.
Softmax function	A generalization the sigmoid func tion. Used to ob classification pr bilities when we have more than two classes.

Figure 1: Activation Functions

1.2 Multi-label vs multi-class

: Multi-class means a classification problem with more than two classes. These classes are distinct, i.e., only one of them could occur at once. e.g. in a corpus of images, we can distinguish b/w dog,cat, and fish. Since sum of all probabilities is 1 here, we should use softmax activation function.

In comparison, multi-label classification means labels which are not distinct and can occur together. For instance, in a chect x-ray scan system, multiple chest diseases such as T.B, cancer, and CoVID can co-exist. Here sum of all probabilities can be greater than one. We will use sigmoid activation function.

1.3 Error Functions

For continuous o/p variable

Root Mean Square Error $E(W,b) = \frac{1}{n}\Sigma_{i=1}^{n}(\hat{y} - y_i)^2$
RMSE penalizes outliers

Absolute Mean Error $E(W,b) = \frac{1}{n}\sum_{t=1}^{n}|\hat{y} - y_i|$

For categorical values

Cross Entropy

$E(W,b) = -\Sigma_{i=1}^{n}\Sigma_{i=1}^{m}\hat{y}_i log(p_i)$

cross-entropy over all training examples
$E(W,b) = -\Sigma_{i=1}^{m}\hat{y}_i log(p_i)$

1.4 Chain Rule

Suppose that we have two functions $f(x)$ and $g(x)$ and they are both differentiable.

1. If we define $F(x) = (f \circ g)(x)$ then the derivative of $F(x)$ is,

$$F'(x) = f'(g(x))\ g'(x)$$

2. If we have $y = f(u)$ and $u = g(x)$ then the derivative of y is,

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

Figure 2: Chain rule explained

(a) $f(x) = \sin(3x^2 + x)$ *Hide Solution* ▾

It looks like the outside function is the sine and the inside function is $3x^2+x$. The derivative is then.

$$f'(x) = \underbrace{\cos}_{\text{derivative of outside function}} \underbrace{(3x^2+x)}_{\text{leave inside function alone}} \underbrace{(6x+1)}_{\text{times derivative of inside function}}$$

Or with a little rewriting,

$$f'(x) = (6x+1)\cos(3x^2+x)$$

(b) $f(t) = (2t^3 + \cos(t))^{50}$ *Hide Solution* ▾

In this case the outside function is the exponent of 50 and the inside function is all the stuff on the inside of the parenthesis. The derivative is then.

$$\begin{aligned} f'(t) &= 50(2t^3+\cos(t))^{49}(6t^2-\sin(t)) \\ &= 50(6t^2-\sin(t))(2t^3+\cos(t))^{49} \end{aligned}$$

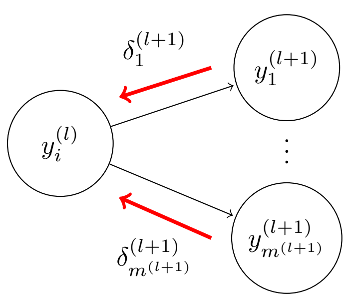
Figure 3: Chain rule examples

1.5 Gradient Descent

Optimization technique to adjust weights
Brute force is not scalable.
Error= —-y_i|
$\Delta W_i = -\alpha \frac{\partial E}{\partial W_i}$
Three variants

1. Batch Gradient Descent: Compute gradient descent over all the points
2. Stochastic Gradient Descent: Select a random point and compute gradient w.r.t that point. Faster than gradient descent but may not reach global minima
3. Mini Batch Gradient Descent: Divide training into mini batches

1.6 Back Propagation



[2][t!] [Backpropagation of errors through the network.]Once evaluated for all output units, the errors $\delta_i^{(L+1)}$ can be propagated backwards..

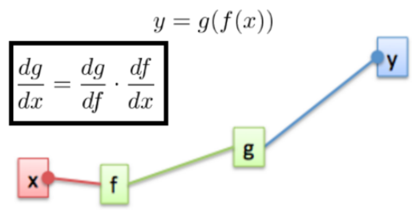


Figure 4: back propagation example