Week 3: Optimization in Neural Networks and Newton's Method:

For a model training summation equation is:

Forward Propagation

$$z_{(i)} = \omega x_{(i)} + b$$

Predicted Output ⇝ $\hat{y}_{(i)} = z_{(i)}$ Loss function ⇝ $L(\omega,b) = \frac{1}{2}(\hat{y}_{(i)} - y_{(i)})^2$

Cost function ⇝ $\mathcal{L}(\omega, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_{(i)} - y_{(i)})$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \omega} = \frac{\partial \mathcal{L}}{\partial \omega} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_{(i)} - y_{(i)}) x_{(i)}$$ chain rule

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = \frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_{(i)} - y_{(i)})$$ chain rule

Backward Propagation

$$\omega = \omega - \alpha \frac{\partial \mathcal{L}}{\partial \omega}$$

$$b = b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

For some complicated perceptrons, there are more inputs with different weights (ω_i).

There are many types of machine learning problems; like:

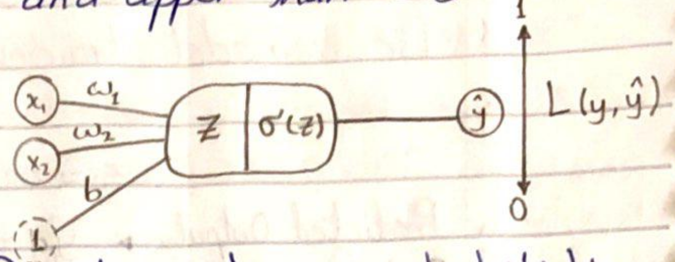
1. Regression problems [as the one shown above]
2. Classification problems [like binary classification]
3. Clustering problems
4. Dimensionality problems
5. Anomaly Detection problems
6. Agent-Environment Interaction problems
7. Recommendation systems problems
8. Generative modeling problems
9. Time series Forecasting problems

Binary classification uses a function on z called sigmoid $(\sigma(z)) = \frac{1}{1+e^{-z}}$, whose derivative $(\sigma'(z)) = \sigma(z)(1-\sigma(z))$.

sigmoid gives outputs between 0 and 1. Lower than 0.5 is a class, and upper than 0.5 is the other class.

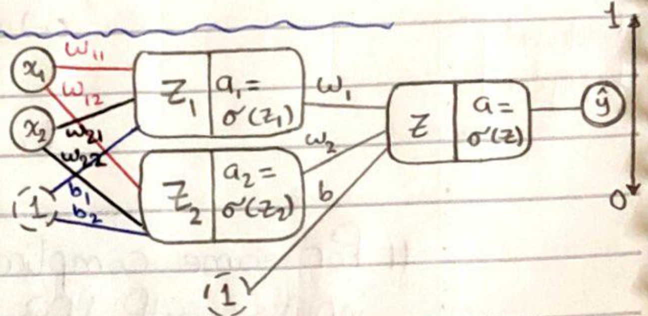
$L \rightarrow$ loss function

$\mathcal{L} \rightarrow$ cost function



→ Because sigmoid functions have probabilistic nature we use log loss for its problems. Then we use gradient descent to get the least loss value from the function (log loss fuction).

Example of a classification neural network:



$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$

$a_1 = \sigma(z_1)$

$z_2 = x_1 w_{12} + x_2 w_{22} + b_2$

$a_2 = \sigma(z_2)$

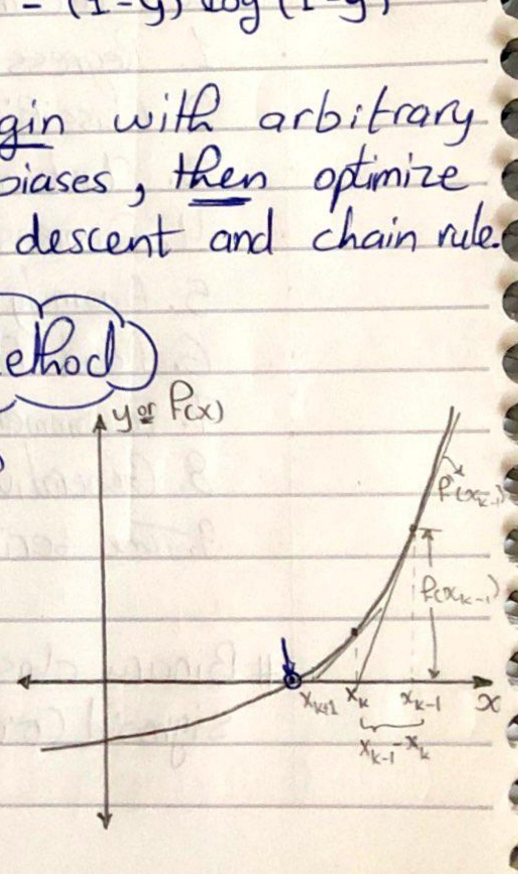
$z = a_1 w_1 + a_2 w_2 + b$

$a = \sigma(z)$ $\quad \hat{y} = a \rightsquigarrow$ (expected output) $\quad y \rightsquigarrow$ real value

$\boxed{\text{Log-loss Function}}$ Loss function $(L(y,\hat{y})) = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$

to solve this example, we begin with arbitrary initial values for weights and biases, then optimize for lower error using gradient descent and chain rule.

Newton's Method



It is a method to find the zeros of a function:

→ taking the tangent at x_{k-1}, then its intersection with x-axis is x_k.

→ repeate the past step till we get close enough to the zero.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Newton's Method for optimization:

Goal: minimize $g(x)$ $\rightsquigarrow$ find zeros of $g'(x)$

step (1): start with some x_0

step (2): $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$; where $f(x) = g'(x)$ & $f'(x) = g''(x)$

$$x_{k+1} = x_k - \frac{g'(x_k)}{g''(x_k)}$$

step(3): Repeate step(2) until you find the candidate(s) for the minimum.

Second Derivative:

Notation $\rightarrow$ Leibniz: $\frac{d^2 f(x)}{dx^2} = \frac{d}{dx}\left(\frac{df(x)}{dx}\right)$

$\rightarrow$ Lagrange: $f''(x)$

It is an indication of the curvature of the graph:

$\rightarrow$ $\frac{d^2 f(x)}{dx^2} > 0$ $\rightsquigarrow$ concave up or convex (has local minimum)

$\rightarrow$ $\frac{d^2 f(x)}{dx^2} < 0$ $\rightsquigarrow$ concave down (has local maximum)

$\rightarrow$ $\frac{d^2 f(x)}{dx^2} = 0$ $\rightsquigarrow$ line or inflection point

$f(x,y) \rightarrow f_x(x,y) \rightarrow f_{xx}(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2}$

$f(x,y) \rightarrow f_x(x,y) \rightarrow f_{xy}(x,y) = \frac{\partial^2 f(x,y)}{\partial x \partial y}$

These two are equal

$f(x,y) \rightarrow f_y(x,y) \rightarrow f_{yx}(x,y) = \frac{\partial^2 f(x,y)}{\partial y \partial x}$

$f(x,y) \rightarrow f_y(x,y) \rightarrow f_{yy}(x,y) = \frac{\partial^2 f(x,y)}{\partial y^2}$

$$\text{Hessian Matrix } (H) = \begin{bmatrix} f_{xx}(x,y) & f_{xy}(x,y) \\ f_{yx}(x,y) & f_{yy}(x,y) \end{bmatrix}$$

	1 Variable	2 Variables
Function	$f(x)$	$f(x,y)$
First Derivative	$f'(x)$: Rate of change of $f(x)$	$f_x(x,y)$: Rate of change w.r.t x $f_y(x,y)$: Rate of change w.r.t y $\nabla f = \begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix}$
Second Derivative	$f''(x)$: Rate of change of the rate of change of $f(x)$	$H(x,y) = \begin{bmatrix} f_{xx}(x,y) & f_{xy}(x,y) \\ f_{yx}(x,y) & f_{yy}(x,y) \end{bmatrix}$

	1 Variable $f(x)$	2 Variables $f(x,y)$	More Variables $f(x_1, x_2, \dots, x_n)$
(Local) Minima	Happy face $f''(x) > 0$	Upper Paraboloid $\lambda_1 > 0$ & $\lambda_2 > 0$	All $\lambda_i > 0$
(Local) Maxima	Sad face $f''(x) < 0$	Down Paraboloid $\lambda_1 < 0$ & $\lambda_2 < 0$	All $\lambda_i < 0$
Need More Information	$f''(x) = 0$	Saddle point $\lambda_1 > 0$ & $\lambda_2 < 0$ $\lambda_1 < 0$ & $\lambda_2 > 0$ Or some $\lambda_i = 0$	Some $\lambda_i > 0$ and Some $\lambda_i < 0$ OR At least one $\lambda_i = 0$

steps to solve Hessians:

step (1): get the gradient of the function (∇f)

step (2): get the hessian of the function (H)

step(3): Calculate $H(0,0) - \lambda I$

$\lambda \rightarrow$ eigenvalue $\quad I \rightarrow$ Identity Matrix

step(4): Get $\det(H(0,0) - \lambda I)$ as a function of λ.

step(5): solve for λ and classify the graph according to the second table.

Newton's Method for 2 variables:

take care of the order of H & ∇f.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \underbrace{H^{-1}(x_k, y_k)}_{(2\times2)\ \text{Matrix}} \cdot \underbrace{\nabla f(x_k, y_k)}_{(2\times1)\ \text{Matrix}}$$

We tend to use gradient descent less than Newton's method, because Newton's method is faster in most of the cases. As the dimensions (variables) gets higher, the more the difference between the two methods becomes obvious.

Real-World datasets are usually linearly inseparable, and there will be a small percentage of errors. More than that, you don't want to build a model that fits too closely (almost exactly) to particular set of data, as it may fail to predict future observations. This problem is known as overfitting.