Model Question Paper-I with effect from 2022(CBCS Scheme)

USN

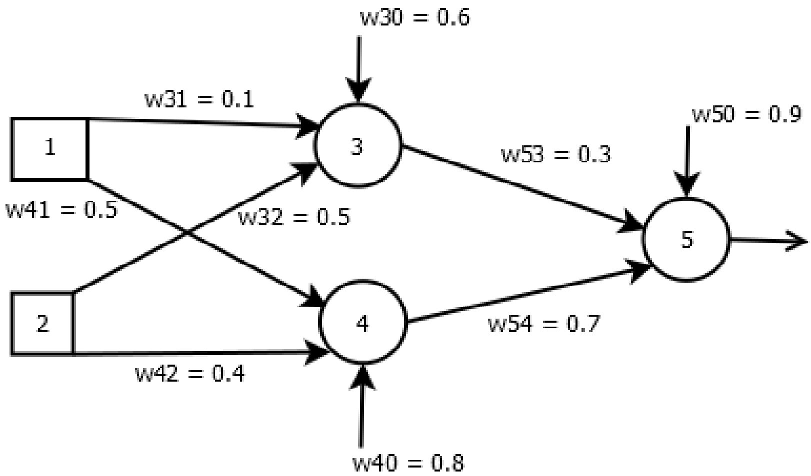
Fourth Semester B.E Degree Examination

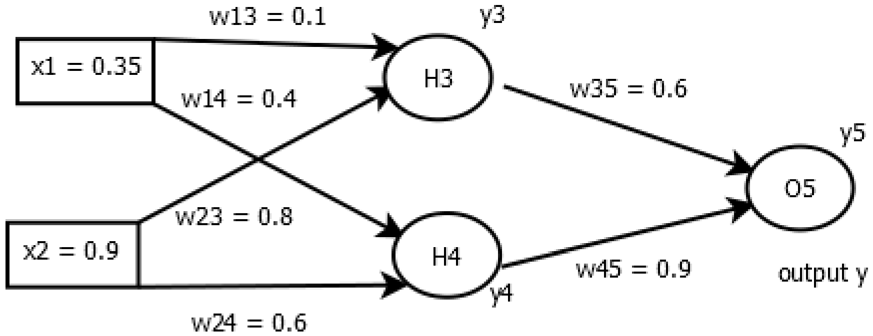
OPTIMIZATION TECHNIQUES (BCS405C)

TIME:03Hours **Max.Marks:100**

Note:

1. Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE**
2. VTU Formula Hand Book is Permitted
3. M: Marks, L: RBT levels, C: Course outcomes.

		Module - 1	M	L	C
Q.1	a	Let $f(x_1, x_2) = e^{x_1 x_2^2}$ where $x_1 = t\cos t$ and $x_2 = t\sin t$ find $\frac{df}{dt}$.	7	L2	CO1
	b	Obtain the gradient of scalar $\phi = 4x_0 + 2x_1 - 3x_2 + x_4$ with respect to the matrix $\vec{x} = \begin{bmatrix} x_0 & x_1 \\ x_2 & x_3 \end{bmatrix}$.	6	L2	CO1
	c	Obtain the power series expansion of $f(x, y) = x^2y + 3y - 2$ in terms of $(x-1)$ and $(y+2)$ up to second degree.	7	L3	CO1
		OR			
Q.2	a	Discuss the gradient of vectors with respect to matrices.	7	L2	CO1
	b	If $\vec{x}, \vec{y} \in \mathbb{R}^2$ and $y_1 = -2x_1 + x_2$, $y_2 = x_1 + x_2$. Show that the Jacobian determinant $\lvert \det \boldsymbol{J} \rvert = 3$.	6	L3	CO1
	c	Find the second order Taylor's series approximation of the function $f(x_1, x_2) = x_1^2 x_2 + 5x_1 e^{x_2}$ about the point $a = 1$, $b = 0$.	7	L3	CO1
		Module – 2			
Q.3	a	Draw a computation graph of the function: $f(x) = \sqrt{x^2 + e^{x^2}} + \cos(x^2 + e^{x^2})$. Also find $\frac{\partial f}{\partial x}$ using automatic differentiation.	8	L3	CO2
	b	Obtain the gradient of quadratic cost.	6	L3	CO2
	c	Find the output at neuron 5, if input vector [0.7, 0.3] using the activation function ReLU. 	6	L3	CO2

		OR			
Q.4	**a**	Let $f(x_1, x_2) = \log(x_1) + x_1 x_2 - \sin(x_2)$. (i) Draw a computational graph of $f(x_1, x_2)$. (ii) Evaluate f at $(x_1, x_2) = (2, 5)$ by forward trace.	8	L3	CO2
	b	Assume that the neuron have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 0.5 and learning rate is 1. Perform another forward pass.  	12	L3	CO2
		Module – 3			
Q.5	**a**	Describe Local and Global optima. List out the differences between Local and Global optima.	5	L2	CO3
	b	Define Hessian matrix. Using the Hessian matrix, classify the relative extreme for the function $f(x, y) = \frac{1}{3}x^3 + xy^2 - 8xy + 3$	7	L3	CO3
	c	Explain the algorithm of sequential search. Using the sequential search, for an array of size 7 with elements 13, 9, 21, 15, 39, 19, and 27 that starts with 0 and ends with size minus one, 6 locate the position of number 39.	8	L3	CO3
		OR			
Q.6	**a**	Minimize $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1 x_2 + x_2^2$ starting from $X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$	7	L3	CO3
	b	Write the algorithm for Fibonacci search method.	6	L2	CO3
	c	Using 3-point interval search method, find $Max\ f(x) = x(5\pi - x)$ on $[0,20]$ with $\varepsilon = 0.1$	7	L3	CO3
		Module – 4			
Q.7	**a**	Use steepest Descent method for f(x, y) = $x_1 - x_2 + 2x_1^2 + 2x_1 x_2 + x_2^2$ starting from the point $x_1 = (0, 0)$	7	L3	CO4
	b	Explain how the Gradient Descent Algorithm works?	6	L2	CO4
	c	Find the Linear Regression Coefficients using Gradient Descent Method.	7	L2	CO4
		OR			
Q.8	**a**	Use the NR method to find the smallest and the second smallest positive roots of the equation tan x = 4x correct to 4 decimal places.	7	L3	CO4
	b	Write the differences between Stochastic Gradient Descent and Mini Batch Gradient Descent methods.	6	L2	CO4
	c	Write the Stochastic Gradient Descent Algorithm.	7	L2	CO4
		Module – 5			

Q.9	a	Explain in brief 1. Adagrad optimization strategy 2. RMSprop	10	L2	CO5
	b	What is the difference between convex optimization and non-convex optimization	5	L2	CO5
	c	Describe the saddle point problem in machine learning	5	L2	CO5
OR					
Q.10	a	Write a short notes on 1.Stochastic gradient descent with momentum 2.ADAM	10	L2	CO5
	b	What is the best optimization algorithm for machine learning	5	L2	CO5
	c	Briefly explain the advantages of RMSprop over Adagrad	5	L2	CO5

Model Question Paper-II with effect from 2022(CBCS Scheme)

USN

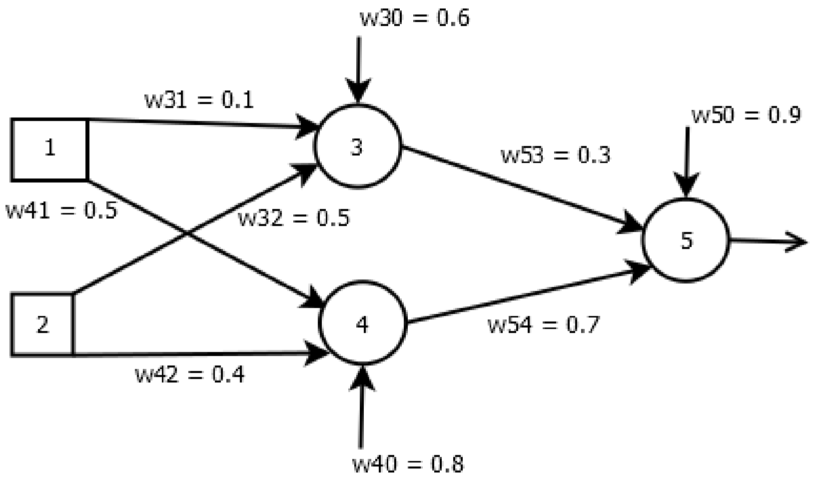
Fourth Semester B.E Degree Examination

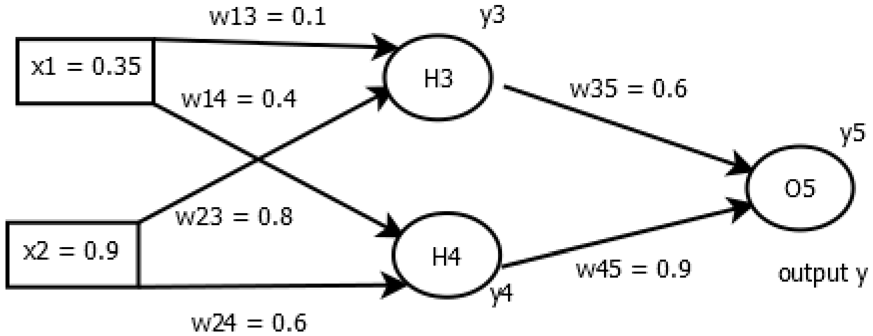
OPTIMIZATION TECHNIQUES (BCS405C)

TIME:03Hours **Max.Marks:100**

Note:

1. Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE**
2. VTU Formula Hand Book is Permitted
3. M: Marks, L: RBT levels, C: Course outcomes.

		Module - 1	M	L	C
Q.1	a	Let $f(x_1, x_2) = x_1^2 + 2x_2$ where $x_1 = \sin t$ and $x_2 = \cos t$ find $\frac{df}{dt}$.	7	L2	CO1
	b	Obtain the gradient of matrix $\vec{f} = \begin{bmatrix} \sin(x_0 + 2x_1) & 2x_1 + x_3 \\ 2x_0 + x_2 & \cos(2x_2 + x_3) \end{bmatrix}$ with respect to the matrix $\vec{x} = \begin{bmatrix} x_0 & x_1 \\ x_2 & x_3 \end{bmatrix}$.	7	L3	CO1
	c	Obtain the partial derivatives for (i) $f(x, y) = (x + 2y^3)^2$ (ii) $f(x, y) = x^2y + xy^3$	6	L3	CO1
		OR			
Q.2	a	Discuss (i) Gradient of a matrix with respect to a vector. (ii) Gradient of a matrix with respect to a matrix.	10	L2	CO1
	b	Find the Taylor's series expansion of the function $f(x, y) = x^2 + 2xy + y^3$ at $(x_0, y_0) = (1, 2)$ up to third degree.	10	L3	CO1
		Module – 2			
Q.3	a	Draw a computation graph of the function: $f(x) = \sqrt{x^2 + e^{x^2}} + \cos(x^2 + e^{x^2})$. Also find $\frac{\partial f}{\partial x}$ using automatic differentiation.	8	L3	CO2
	b	Obtain the gradient of quadratic cost.	6	L3	CO2
	c	Find the output at neuron 5, if input vector [0.7, 0.3] using the activation function ReLU. 	6	L3	CO2

		OR			
Q.4	a	Let $f(x_1, x_2) = \log(x_1) + x_1x_2 - \sin(x_2)$. (i) Draw a computational graph of $f(x_1, x_2)$. (ii) Evaluate f at $(x_1, x_2) = (2, 5)$ by forward trace.	8	L3	CO2
	b	Assume that the neuron have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 0.5 and learning rate is 1. Perform another forward pass.  	12	L3	CO2
		Module – 3			
Q.5	a	Describe Local and Global optima. List out the differences between Local and Global optima.	5	L2	CO3
	b	Define Hessian matrix. Using the Hessian matrix, classify the relative extreme for the function $f(x, y) = \frac{1}{3}x^3 + xy^2 - 8xy + 3$	7	L3	CO3
	c	Explain the algorithm of sequential search. Using the sequential search, for an array of size 7 with elements 13, 9, 21, 15, 39, 19, and 27 that starts with 0 and ends with size minus one, 6 locate the position of number 39.	8	L2	CO3
		OR			
Q.6	a	Minimize $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$ starting from $X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$	8	L3	CO3
	b	Write the algorithm for Fibonacci search method.	6	L3	CO3
	c	Using 3-point interval search method, find $Max\ f(x) = x(5\pi - x)$ on $[0,20]$ with $\varepsilon = 0.1$	6	L2	CO3
		Module – 4			
Q.7	a	Use steepest Descent method for f(x, y) = $x_1 – x_2 + 2\,x_1^2 + 2x_1x_2 + x_2^2$ starting from the point $x_1 = (0, 0)$	7	L2	CO4
	b	Explain how the Gradient Descent Algorithm works?	6	L2	CO4
	c	Find the Linear Regression Coefficients using Gradient Descent Method.	7	L2	CO4
		OR			
Q.8	a	Use the NR method to find the smallest and the second smallest positive roots of the equation tan x = 4x correct to 4 decimal places.	7	L2	CO4
	b	Write the differences between Stochastic Gradient Descent and Mini Batch Gradient Descent methods.	7	L2	CO4
	c	Write the Stochastic Gradient Descent Algorithm.	6	L2	CO4
		Module – 5			

Q.9	a	Explain in brief 1. Adagrad optimization strategy 2. RMSprop	10	L2	CO5
	b	What is the difference between convex optimization and non-convex optimization	5	L2	CO5
	c	Describe the saddle point problem in machine learning	5	L2	CO5
		OR			
Q.10	a	Write a short notes on 1.Stochastic gradient descent with momentum 2.ADAM	10	L2	CO5
	b	What is the best optimization algorithm for machine learning	5	L2	CO5
	c	Briefly explain the advantages of RMSprop over Adagrad	5	L2	CO5

MQP I

Q1 a) $f(x_1, x_2) = e^{x_1 x_2^2}$ where $x_1 = t\cos t$

$x_2 = t\sin t$

$$\frac{df}{dt} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}$$

$$= e^{x_1 x_2^2}(1 \cdot x_2^2) \cdot (1 \cdot \cos t + t(-\sin t))$$

$$+ e^{x_1 x_2^2}(x_1(2x_2))(1 \cdot \sin t + t(\cos t))$$

$$= e^{x_1 x_2^2}\left[x_2^2(\cos t - t\sin t) + 2x_1x_2(\sin t + t\cos t)\right]$$

$$= e^{t\cos t(t^2\sin^2 t)}\left[t^2\sin^2 t(\cos t - t\sin t) + 2t\cos t\, t\sin t(\sin t + t\cos t)\right]$$

$$= e^{t^3\sin^2 t\cos t}\left[t^2\sin^2 t\cos t - t^3\sin^3 t + 2t^2\sin^2 t\cos t + 2t^3\cos^2 t\sin t\right]$$

$$= e^{t^3\sin^2 t\cos t}\left[3t^2\sin^2 t\cos t - t^3\sin^3 t + 2t^3\sin t\cos^2 t\right]$$

MQP I

Q1 b) $\phi = 4x_0 + 2x_1 - 3x_2 + x_4 \quad \vec{x} = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix}$

$$\nabla \times \phi = \begin{pmatrix} \frac{\partial \phi}{\partial x_0} & \frac{\partial \phi}{\partial x_1} \\ \frac{\partial \phi}{\partial x_2} & \frac{\partial \phi}{\partial x_3} \end{pmatrix}$$

$$= \begin{pmatrix} 4 & 2 \\ -3 & 1 \end{pmatrix}$$

Q1 c) $f(x,y) = x^2y + 3y - 2$

$$f(x,y) = f(a,b) + \frac{1}{1!}\left[f_x(a,b)(x-a) + f_y(a,b)(y-b)\right]$$

$$+ \frac{1}{2!}\left[f_{xx}(a,b)(x-a)^2 + 2(x-a)(y-b)f_{xy}(a,b) + f_{yy}(a,b)(y-b)^2\right] + \cdots$$

$a = 1 \quad b = -2$

$$f(x,y) = f(1,-2) + \frac{1}{1!}\left[(x-1)f_x(1,-2) + (y+2)f_y(1,-2)\right]$$

$$+ \frac{1}{2!}\left[(x-1)^2 f_{xx}(1,-2) + 2(x-1)(y+2)f_{xy}(1,-2) + (y+2)^2 f_{yy}(1,-2)\right]$$

$f(x,y) = x^2y + 3y - 2, \quad f_x = 2xy, \quad f_y = x^2 + 3$

$f_{xx} = 2y \quad f_{yy} = 0 \quad f_{xy} = \frac{\partial}{\partial x}(f_y) = \frac{\partial}{\partial x}(x^2+3) = 2x$

$f(1,-2) = 1^2(-2) + 3(-2) - 2 = -10$

$f_y(1,-2) = 1^2 + 3 = 4$

$f_x(1,-2) = 2(1)(-2) = -4$

$f_{yy}(1,-2) = 0$

$f_{xx}(1,-2) = -4 \quad f_{xy}(1,-2) = 2$

$$f(x,y) = -10 + \frac{1}{1!}\left[(x-1)(-4) + (y+2)(4)\right]$$
$$+ \frac{1}{2!}\left[-4(x-1)^2 + 2(x-1)(y+2)(2) + (y+2)^2(0)\right] + \cdots$$

$$= -10 + \frac{1}{1!}\left[-4(x-1) + 4(y+2)\right]$$
$$+ \frac{1}{2!}\left[-4(x-1)^2 + 4(x-1)(y+2)\right]$$

Q2 b) $\vec{x}, \vec{y} \in \mathbb{R}^2$ $y_1 = -2x_1 + x_2$, $y_2 = x_1 + x_2$

$$J = \begin{pmatrix} \partial y_1/\partial x_1 & \partial y_1/\partial x_2 \\ \partial y_2/\partial x_1 & \partial y_2/\partial x_2 \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ 1 & 1 \end{pmatrix}$$

$$|J| = |-2-1| = 3$$

Q2 c) $f(x_1, x_2) = x_1^2 x_2^3 + 5x_1 e^{x_2}$ abt the point $a=1,\ b=0$

$$f(x,y) = f(a,b) + \frac{1}{1!}\left[(x-a) f_x(a,b) + (y-b) f_y(a,b)\right]$$
$$+ \frac{1}{2!}\left[(x-a)^2 f_{xx}(a,b) + 2(x-a)(y-b) f_{xy}(a,b) + f_{yy}(a,b)(y-b)^2\right] + \cdots$$

$$f(x,y) = f(1,0) + \frac{1}{1!}\left[(x-1) f_x(1,0) + (y-0) f_y(1,0)\right]$$
$$+ \frac{1}{2!}\left[(x-1)^2 f_{xx}(1,0) + 2(x-1)(y-0) f_{xy}(1,0) + f_{yy}(1,0)(y-0)^2\right] + \cdots$$

$$f(x_1,x_2) = f(1,0) + \frac{1}{1!}\left[(x_1-1) f_{x_1}(1,0) + (x_2-0) f_{x_2}(1,0)\right]$$
$$+ \frac{1}{2!}\left[(x-1)^2 f_{x_1x_1}(1,0) + 2(x_1-1)(x_2-0) f_{x_1x_2}(1,0) + f_{x_2x_2}(1,0)(x_2-0)^2\right] + \cdots$$

$$f(x_1, x_2) = x_1^2 x_2 + 5x_1 e^{x_2}$$
$$f(1,0) = 1^2(0) + 5(1)e^0 = 5$$

$$f_{x_1} = 2x_1x_2 + 5e^{x_2} \qquad f_{x_2} = x_1^2 + 5x_1 e^{x_2}$$

$$f_{x_2x_2} = 5x_1 e^{x_2}$$

$$f_{x_1x_1} = 2x_1$$

$$f_{x_1x_2} = \frac{\partial}{\partial x_1}\left(x_1^2 + 5x_1 e^{x_2}\right) = 2x_1 + 5e^{x_2}$$

$$f_{x_1}(1,0) = 2(1)(0) + 5e^0 = 5$$

$$f_{x_2}(1,0) = 1^2 + 5(1)e^0 = 6$$

$$f_{x_1 x_1} = 2(1) = 2 \qquad f_{x_2 x_2} = 5(1)e^0 = 5$$

$$f_{x_1 x_2}(1,0) = 1^2(0) + 5(1)e^0 = 5$$

$$f(x_1, x_2) = 5 + \frac{1}{1!}\left[(x_1 - 1)\,5 + 6\,(\cancel{y}x_2 - 0)\right]$$

$$+ \frac{1}{2!}\left[2(x_1-1)^2 + 2(5)(x_1-1)(\cancel{y}_2 - 0) + 5(x_2-0)^2\right]$$

$$= 5 + \frac{1}{1!}\left[5(x_1-1) + 6x_2\right]$$

$$+ \frac{1}{2!}\left[2(x_1-1)^2 + 10(x_1-1)x_2 + 5x_2^2\right] + \cdots$$

MQP II

Q1. a) Let $f(x_1, x_2) = x_1^2 + 2x_2$ where $x_1 = \sin t$
$x_2 = \cos t$

find $\frac{df}{dt}$

$$\frac{df}{dt} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}$$

$$= (2x_1)(\cos t) + 2(-\sin t)$$

$$= 2\sin t \cos t - 2\sin t = 2\sin t(\cos t - 1)$$

Q1 c) Find the partial derivatives of

i) $f(x,y) = (x+2y^3)^2$ ii) $f(x,y) = x^2y + xy^3$

i) $\frac{\partial f(x,y)}{\partial x} = 2(x+2y^3) \cdot \frac{\partial}{\partial x}(x+2y^3) = 2(x+2y^3)$

$\frac{\partial}{\partial y} f(x,y) = 2(x+2y^3)\frac{\partial}{\partial y}(x+2y^3)$

$= 2(x+2y^3)(6y^2) = 12(x+2y^3)y^2$

ii) $\frac{\partial}{\partial x} f(x,y) = 2xy + y^3$ $\frac{\partial}{\partial y}(f(x,y))$

$= x^2 + 3xy^2$

Map II

Q1 b) $\vec{f} = \begin{pmatrix} \sin(x_0 + 2x_1) & 2x_1 + x_3 \\ 2x_0 + x_2 & \cos(2x_2 + x_3) \end{pmatrix}$

$$\vec{x} = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix}$$

$$\frac{\partial \vec{y}}{\partial x} = \begin{pmatrix} \begin{pmatrix} \cos(x_0+2x_1) & 0 \\ 2 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 1 & -2\sin(2x_2+x_3) \end{pmatrix} \\ \begin{pmatrix} 2\cos(x_0+2x_1) & 2 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 0 & -\sin(2x_2+x_3) \end{pmatrix} \end{pmatrix}$$

MQP II

Q2 b) Find the Taylor's series of $f(x,y) = x^2 + 2xy + y^3$ at $(x_0, y_0) = (1,2)$ upto the 3rd degree.

Ans $f(x,y) = f(a,b) + \frac{1}{1!}\left[(x-a) f_x(a,b) + (y-b) f_y(a,b)\right]$

$+ \frac{1}{2!}\left[(x-a)^2 f_{xx}(a,b) + 2(x-a)(y-b) f_{xy}(a,b) + (y-b)^2 f_{yy}(a,b)\right] +$

$\frac{1}{3!}\left[(x-a)^3 f_{xxx}(a,b) + 3(x-a)^2(y-b) f_{xxy}(a,b) + (y-b)^3 f_{yyy}(a,b)\right] + \ldots$ (1)

$+ 3(x-a)(y-b)^2 f_{xyy}(a,b)$

$a = 1 \quad b = 2 \quad f_x = 2x + 2y \quad f_x(1,2) = 2(1) + 2(2) = 6$

$f_y = 2x + 3y^2 \quad f_y(1,2) = 2(1) + 3(2^2) = 14$

$\nabla_{x,y} f(1,2) = [6, 14] \in \mathbb{R}^{1\times 2}$

$D_{x,y} f(1,2) = [6 \quad 14] \begin{bmatrix} x-1 \\ y-2 \end{bmatrix} = 6(x-1) + 14(y-2)$ (2)

$\dfrac{D_{x,y} f(1,2)}{1!} \delta$

$f_{xx} = 2 \quad f_{yy} = 6y \quad f_{yx} = 2 \quad f_{xy} = 2$

at $(1,2)$ $f_{xx} = 2 \quad f_{yy} = 12 \quad f_{yx} = 2 \quad f_{xy} = 2$

$H = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 2 & 12 \end{pmatrix}$

$\dfrac{D^2_{x,y} f(1,2)}{2!} \delta^2 = \frac{1}{2} \delta^T H(1,2) \delta$

$$= \frac{1}{2}\begin{bmatrix} x-1 & y-2 \end{bmatrix}\begin{bmatrix} 2 & 2 \\ 2 & 12 \end{bmatrix}\begin{bmatrix} x-1 \\ y-2 \end{bmatrix}$$

$$= (x-1)^2 + 2(x-1)(y-2) + 6(y-2)^2 \quad — \quad (3)$$

3rd order partial derivatives

$f_{xxx} = 0 \quad f_{yyy} = 6 \quad f_{xxy} = 0 \quad f_{xyy} = 0$

$f_{yxx} = 0 \quad f_{yyx} = 0$

$$D^3_{x,y} f = \begin{bmatrix} \frac{\partial H}{\partial x} & \frac{\partial H}{\partial y} \end{bmatrix} \in \mathbb{R}^{2\times 2\times 2}$$

$$D^3_{x,y} f[:\ :\ 1] = \frac{\partial H}{\partial x} = \begin{bmatrix} \frac{\partial^3 f}{\partial x^3} & \frac{\partial^3 f}{\partial x^2 \partial y} \\ \frac{\partial^3 f}{\partial x \partial y \partial x} & \frac{\partial^3 f}{\partial x \partial y^2} \end{bmatrix}$$

$$D^3_{x,y} f[:\ :\ 2] = \frac{\partial H}{\partial y} = \begin{bmatrix} \frac{\partial^3 f}{\partial y \partial x^2} & \frac{\partial^3 f}{\partial y \partial x \partial y} \\ \frac{\partial^3 f}{\partial y^2 \partial x} & \frac{\partial^3 f}{\partial y^3} \end{bmatrix}$$

$$D^3_{x,y} f[:\ :\ 1] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$D^3_{x,y} f[:\ :\ 2] = \begin{bmatrix} 0 & 0 \\ 0 & 6 \end{bmatrix}$$

$$\frac{D^3_{x,y} f(1,2)}{3!} \quad \delta^3 = (y-2)^3$$

(1) is

$$f(x,y) = f(1,2) + \frac{1}{1!}\left[(x-1) f_x(1,2) + (y-2) f_y(1,2)\right]$$
$$+ \frac{1}{2!}\left[(x-1)^2 f_{xx}(1,2) + (2(x-1)(y-2) f_{xy}(1,2) + (y-2)^2 f_{yy}(1,2)\right]$$
$$+ \frac{1}{3!}\left[(x-1)^3 f_{xxx}(1,2) + 3(x-1)^2(y-2) f_{xxy}(1,2) + 3(x-1)(y-2)^2 f_{xyy}(1,2) + (y-2)^3 f_{yyy}(1,2)\right] + \cdots$$

$$f(x,y) = f(1,2) + \frac{1}{1!}\left[(x-1)6 + 14(y-2)\right]$$

$$+ \frac{1}{2!}\left[(x-1)^2 2 + 2\cdot 2(x-1)(y-2) + 12(y-2)^2\right]$$

$$+ \frac{1}{3!}\left[0.(x-1)^3 + 3.0(x-1)^2(y-2) + 3.0.(x-1)(y-2)^2 + 6(y-2)^3\right] + \cdots$$

$$f(1,2) = 1 + 2(1)(2) + 2^3 = 13$$

$$f(x,y) = 13 + \frac{1}{1!}\left[6(x-1) + 14(y-2)\right]$$

$$+ \frac{1}{2!}\left[2(x-1)^2 + 4(x-1)(y-2) + 12(y-2)^2\right]$$

$$+ \frac{1}{3!}\left[6(y-2)^3\right] + \cdots$$

$$= 13 + 6(x-1) + 14(y-2) + (x-1)^2 + 2(x-1)(y-2) + 6(y-2)^2 + (y-2)^3 + \cdots$$

Q3 a) $f(x) = \sqrt{x^2 + e^{x^2}} + \cos(x^2 + e^{x^2})$

Ans. Introduce intermediate variables

$a = x^2$, $b = exp(a)$, $c = a + b$, $d = \sqrt{c}$ $e = \cos(c)$

$f = d + e$

$\frac{\partial a}{\partial x} = 2x$; $\frac{\partial b}{\partial a} = exp(a)$; $\frac{\partial c}{\partial a} = 1 = \frac{\partial c}{\partial b}$

$\frac{\partial d}{\partial c} = \frac{1}{2\sqrt{c}}$ $\frac{\partial e}{\partial c} = -\sin(c)$ $\frac{\partial f}{\partial d} = 1 = \frac{\partial f}{\partial e}$

Compute $\partial f / \partial x$ from the output by working backward

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c}$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} ; \frac{\partial f}{\partial a} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} ;$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x}$$

$$\frac{\partial f}{\partial c} = 1 \cdot \frac{1}{2\sqrt{c}} + 1(-\sin(c))$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \cdot 1$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \cdot \cancel{1}$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} exp(a) + \frac{\partial f}{\partial c} \cdot 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \cdot 2x$$

Q2 b) ~~Compute~~
With a single data point, we can use quadratic cost alone. It is defined as

$$C = (\hat{y} - y)^2$$

Calculate gradient of C w.r.to parameters

$$C = u^2 \quad \text{where} \quad u = \hat{y} - y$$

$$\frac{dC}{du} = \frac{\partial C}{\partial u} = 2u = 2(\hat{y} - y)$$

$$\frac{\partial u}{\partial \hat{y}} = 1 - 0 = 1 \qquad \frac{\partial C}{\partial \hat{y}} = \frac{\partial C}{\partial u}\frac{\partial u}{\partial \hat{y}}$$

$$= 2(\hat{y} - y)(1) = 2(\hat{y} - y)$$

$$= 2(\hat{y} - y)$$

$$\hat{y} = mx + b$$

$$\frac{\partial \hat{y}}{\partial b} = 0 + 1 = 1 \qquad \frac{\partial \hat{y}}{\partial m} = 1(x) + 0 = x$$

$$\frac{\partial C}{\partial \hat{y}} = 2(\hat{y} - y) \qquad \frac{\partial C}{\partial m} = \frac{\partial C}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial m}$$

$$= (2(\hat{y} - y))x = 2x(\hat{y} - y)$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b} = 2(\hat{y} - y)$$

$$\frac{\partial C}{\partial m} = 2x(\hat{y} - y)$$

· Gradient of cost ∇C

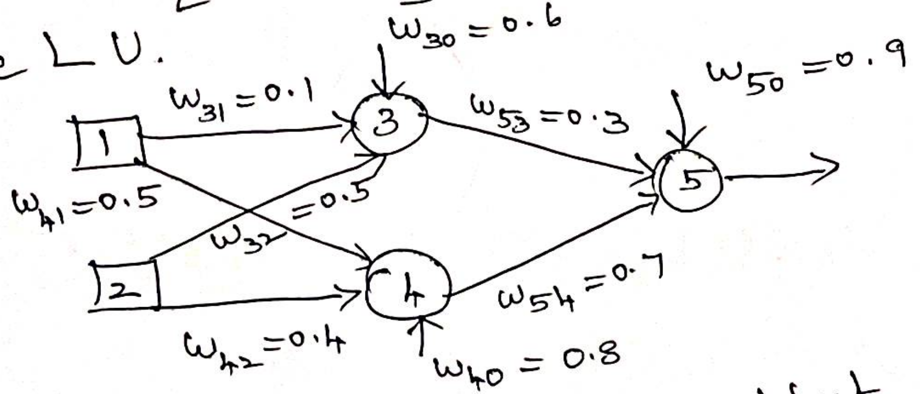
$$\nabla C = \nabla_p C = \left[\frac{\partial C}{\partial p_1} \; \frac{\partial C}{\partial p_2} \; \cdots \; \frac{\partial C}{\partial p_m}\right]^T$$

There are only 2 parameters b and m:

$$\nabla C = \left[\frac{\partial C}{\partial b} \; \frac{\partial C}{\partial m}\right]^T$$

The Gradient of cost ∇C is a vector of all the partial derivatives of C w.r.to each of the individual model parameters.

Q3 c) Find the output at neuron 5, if input vector [0.7 0.3] using the activation function ReLU.



Forward Pass Compute output for y_3 y_4 y_5

$$a_j = \sum_j (w_{i,j} + x_j) \qquad y_j = f(a_j) = \frac{1}{1+e^{-a_j}}$$

What would be the output at Neuron 5 if input vector is [0.7 0.3] and the activation func is logistic function?

Weighted sum calculation for Neuron 3 is

$$z_3 = (0.6)(1) + (0.1)(0.7) + (0.5)(0.3) = 0.82$$
$$= 0.60 + 0.07 + 0.15$$

III'y for Neuron 4 $z_4 = (0.8)(1) + (0.2)(0.7) + (0.4)(0.3)$
$= 1.06$

Using a logistic activation function the activation for Neuron 3 is $a_3 = \text{logistic}(z_3) = \frac{1}{1+e^{-0.82}}$

$= 0.6942$

III'y for Neuron 4, the activation is

$$a_4 = \text{logistic}(z_4) = \frac{1}{1+e^{-1.06}} = 0.7427$$

Weighted sum calculation for Neuron 5 can now be calculated as

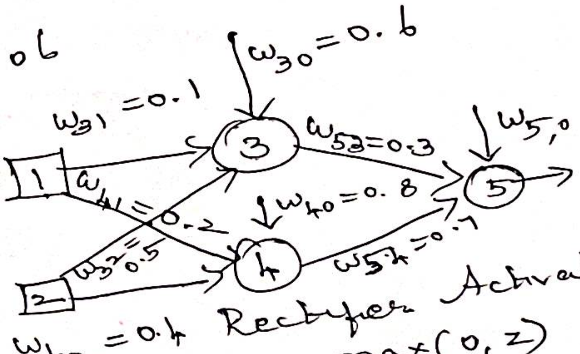
$$z_5 = (0.9)(1) + (0.3)(0.6942) + (0.7)(0.7427)$$
$$= 1.6282$$

$$a_5 = \text{logistic}(z_5) = \frac{1}{1+e^{-1.6282}} = 0.8359$$

What would be the output at Neuron 5 if input vector is [0.7 0.3] and the activation function is ReLU function?

$z_3 = 0.82$ $z_4 = 1.06$

$a_3 = 0.82$ $a_4 = 1.06$



Rectifier Activation function
max(0, z)

Weighted sum calculation for Neuron 5

$$z_5 = (0.9)(1) + (0.3)(\underset{a_3}{0.82}) + (0.7)(\underset{a_4}{1.06}) = 1.888$$

Rectifier Activation function max(0, z)

M&I Q4 a)

Let $f(x, y) = \log x + xy - \sin y$

$f(x_1, x_2) = \log x_1 + x_1 x_2 - \sin(x_2)$

i) Draw a computational graph of $f(x_1, x_2)$

ii) Evaluate f at $(x_1, x_2) = (2, 5)$ by forward trace

Computational Graph

1. Input nodes
   - x_1
   - x_2
2. Intermediate Computations:
   - $\log(x_1)$ (denoted as a)
   - $x_1 x_2$ (denoted as b)
   - $\sin(x_2)$ (denoted as c)
3. Final Computation
   - $a + b - c$

Evaluation at $(x, y) = (2, 5)$

1. Input values:
   - $x = 2$
   - $y = 5$
2. Intermediate Computations
   - $a = \log(2)$
   - $b = 2 \times 5 = 10$
   - $c = \sin(5)$
3. Final Computation:
   - $f(x, y) = a + b - c = \log(2) + 10 - \sin(5)$

Step by Step Evaluation

1. Compute $a = \log(2)$:
   $a = \log(2) \approx 0.6931$

2. Compute $b = 2 \times 5$:
   $b = 10$

3. Compute $c = \sin(5)$:
   $c \approx -0.9589$

4. Combine the results to get $f(x,y)$
   $f(2,5) = 0.6931 + 10 - (-0.9589) = 0.6931 + 10 + 0.9589 \approx 11.6520$

Value of the function at $(x,y) = (2,5)$ is 11.6520

Computational Graph Diagram

$x \rightarrow (2) \rightarrow [\log] \rightarrow (a = \log(x)) \rightarrow 0.6931 \rightarrow$

$[+] \rightarrow (a+b-c) \rightarrow (11.6520)$

$y \rightarrow (5) \rightarrow [*] \rightarrow (b = xy) \rightarrow (10)$

$\rightarrow [\sin] \rightarrow (c = \sin(y)) \rightarrow (-0.9589)$

$x \rightarrow (2) \rightarrow \log \rightarrow (a = \log(x)) \rightarrow (0.6931) \rightarrow$

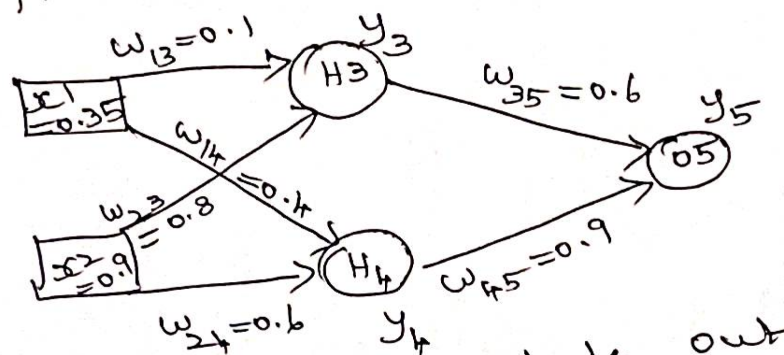
$\rightarrow [+] \rightarrow (a+b-c) \rightarrow (11.6520)$

$y \rightarrow (5) \rightarrow [*] \rightarrow (b = xy) \rightarrow (10) \rightarrow$

$\rightarrow [\sin] \rightarrow (c = \sin(y)) \rightarrow (-0.9589)$

This diagram shows how each part of the function is computed and combined to evaluate $f(2,5)$.

Q4 b)



Forward Pass Compute output for y_3, y_4 and y_5

$a_j = \sum(w_{ij} * x_i)$ $\quad y_j = f(a_j) = \frac{1}{1+e^{-a_j}}$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2) = \underline{0.755}$

$= (0.1)(0.35) + (0.8)(0.9)$

$y_3 = f(a_1) = \frac{1}{1+e^{-0.755}} = \underline{0.68}$

$a_2 = (w_{14} * x_1) + (w_{24} * x_2) = 0.68$

$= (0.4 * 0.35) + (0.6 * 0.9)$

$y_4 = f(a_2) = \frac{1}{1+e^{-0.68}} = \underline{0.6637}$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4) = 0.801$

$= (0.3 * 0.68) + (0.9 * 0.6637)$

$y_5 = f(a_3) = \frac{1}{1+e^{-0.801}} = \underline{0.69}$ (Network output)

Each weight changed by

$\Delta w_{ji} = \eta \delta_j o_i$

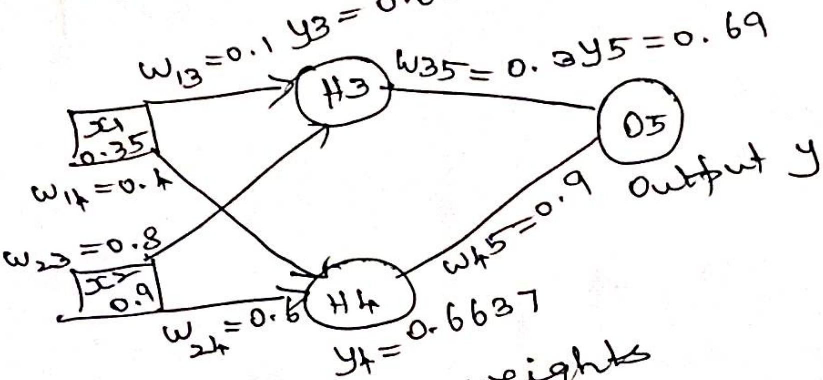
$\delta_j = o_j(1-o_j)(t_j - o_j)$ if j is an output unit

$\delta_j = o_j(1-o_j)(\sum_k \delta_k w_{kj})$ if j is a hidden unit

where η is a constant called the learning rate

t_j is the correct teacher output for unit j

δ_j is the error measure for unit j



Compute new weights

$\Delta w_{ji} = \eta \, \delta_j O_i$

$\Delta w_{45} = \eta \, \delta_5 y_4 = 1(-0.0406)(0.6637) = -0.0269$

$w_{45}(new) = \Delta w_{45} + w_{45}(old) = -0.0269 + 0.9 = 0.8731$

$\Delta w_{14} = \eta \, \delta_4 x_1 = 1(-0.0082)(0.35) = -0.00287$

$w_{14}(new) = \Delta w_{14} + w_{14}(old) = -0.00287 + 0.4 = 0.3971$

For output unit

$\delta_5 = y(1-y)(y_{target} - y)$
$= 0.69 * (1-0.69) * (0.5-0.69) = -0.0406$

For hidden unit

$\delta_3 = y_3(1-y_3) \, w_{35} * \delta_5$
$= 0.68 * (1-0.68) * (0.3 * -0.0406) = -0.00265$

$\delta_4 = y_4(1-y_4) \, w_{45} * \delta_5$
$= 0.6637 * (1-0.6637) * (0.9 * -0.0406) = 0.002$

Similarly update all other weights

i	j	w_{ij}	δ_j	x_i	η	Update w_{ij}
1	3	0.1	-0.00265	0.35	1	0.0991
2	3	0.8	-0.00265	0.9	1	0.7976
1	4	0.4	-0.0082	0.35	1	0.3971
2	4	0.6	-0.0082	0.9	1	0.5926
3	5	0.3	-0.0406	0.68	1	0.2724
4	5	0.9	-0.0406	0.6637	1	0.8731

May

Q5 b) Find the relative extrema of the function
$f(x,y) = \frac{1}{3}x^3 + xy^2 - 8xy + 3$

Ans $f_x = x^2 + y^2 - 8y$ $\quad f_y = 2xy - 8x$

Set $f_x = 0$ $\quad f_y = 0$

$x^2 + y^2 - 8y = 0$ — ① $\quad 2xy - 8x = 0$ — ②

$2x(y-4) = 0$

$\Rightarrow x = 0,\ y = 4$

Case 1 $x = 0$

① is $y^2 - 8y = 0 \Rightarrow y(y-8) = 0 \Rightarrow y = 0, 8$

We have 2 critical points $(0,0)$ $(0,8)$

Case 2 $y = 4$ ① is $x^2 + 16 - 32 = 0 \Rightarrow x = \pm 4$

We have 2 more critical points $(4,4)$ + $(-4,4)$

Step 2 $H = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{pmatrix} = \begin{pmatrix} 2x & 2y-8 \\ 2y-8 & 2x \end{pmatrix}$ — ③

Evaluate H at critical points

$H(0,0) = \begin{pmatrix} 0 & -8 \\ -8 & 0 \end{pmatrix}$ Det $= -64 < 0$ indicating a saddle point

$H(0,8) = \begin{pmatrix} 0 & 8 \\ 8 & 0 \end{pmatrix}$ Det $= -64$ indicating a saddle point

$H(4,4) = \begin{pmatrix} 8 & 0 \\ 0 & 8 \end{pmatrix}$ Det $= 64$ $f_{xx} = 8 > 0$ indicating a local minimum

$H(-4,4) = \begin{pmatrix} -8 & 0 \\ 0 & -8 \end{pmatrix}$ Det $= 64$ $f_{xx} = -8 < 0$ indicating a local max

Ans $(0,0)$ saddle point
$(0,8)$ saddle point
$(4,4)$ local minimum $\quad (-4,4)$ local maximum

Q5 a) A local max/min occurs at $x = c$ when $f(c) > f(x)$ for x values around c.

A global max/min occurs at $x = c$ if $f(x) \leq f(c)$ for all values of x in the domain.

It is possible to have several global max/min if the function reaches its peak value at more than one point.

Q5 b) The Hessian matrix is a square matrix that represents the 2nd order partial derivatives of a scalar valued function. It provides information about the local curvature of the function and is used in optimisation and analysis of functions to determine concavity or convexity.

For a function $f: R^n \rightarrow R$ that is twice continuously differentiable, the Hessian matrix is defined as

$$H(f) = \begin{bmatrix} \partial^2 f/\partial x_1^2 & \partial^2 f/\partial x_1 \partial x_2 & \cdots & \partial^2 f/\partial x_1 \partial x_n \\ \partial^2 f/\partial x_2 \partial x_1 & \partial^2 f/\partial x_2^2 & \cdots & \partial^2 f/\partial x_2 \partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial^2 f/\partial x_n \partial x_1 & \partial^2 f/\partial x_n \partial x_2 & \cdots & \partial^2 f/\partial x_n^2 \end{bmatrix}$$

To locate the position of the number 39 in the array [13, 9, 21, 15, 39, 19, 27] using sequential search we can iterate through the array from the 1st element to the last. We perform the search step by step

1. Start at index 0.
2. Check if the element at the current index is equal to 39.
3. If it is return the current index as the position
4. If it is not, move to the next index.
5. Repeat steps 2-4 until the element is found or the end of the array is reached

Let's write out this process:

1. Check index 0: the value is 13, it's not 39
2. Check index 1: the value is 9 it's not 39
3. Check index 2: the value is 21 "
4. check index 3 : the value is 15 "
5. Check index 4 : the value is 39. This is a match.

The position of the number 39 in the array is 4.

So the Sequential search finds the number 39 at index 4 in the array.

Q6c) Three point Search

This method is used for unconstraint optimisation. In this method we divide the interval into 4 equal parts. We select the central point as functional value which contained max/min. Then we select its interval around its central functional value. We expect this process until we reach the value that shows less value as ϵ tolerance

$$|f(x_c) - f(x_{cnm})| < \epsilon$$

called stopping criteria.

The above step is interval search to find

Q By using 3 point interval search to find max $f(x) = x(5\pi - x)$ in $[0, 20]$ with $\epsilon = 0.1$

Ans $f(x) = 5\pi x - x^2$ $\quad n = \frac{20-0}{4} = 5$

	x	$f(x)$
a	0	0
x_0	5	53.54
x_1	10	57.08
x_2	15	10.62
b	20	-85.84

Centre is x_1 SI $[5, 15]$

max value at $x_1 = 10$

$|57.08 - 53.54| = 3.54 \nless \epsilon$

Iteration 1

$$f(x) = 5\pi x - x^2 \qquad n = \frac{15-5}{4} = 2.5$$
$$[5, 15]$$

x_0	5	53.54 ✓
x_3	7.5	61.56
x_1	10	57.08 ✓
x_4	12.5	40.1
x_2	15	10.62

Centre is at x_3 $SI = [5, 10]$

max value at $x_3 = 7.5$

$$|f(x_3) - f(x_1)| = |61.56 - 57.08| = 4.48 \nless \epsilon$$

Iteration 2

$$f(x) = 5\pi x - x^2 \qquad n = \frac{10-5}{4} = 1.25$$

x_0	5	53.54
x_5	6.25	59.8 ✓
x_3	7.5	61.56
x_6	8.75	60.88 ✓
x_1	10	57.08

Centre is at x_3 SI $[6.25, 8.75]$

$$|f(x_3) - f(x_6)| = |61.56 - 60.88| = 0.68 \nless \epsilon$$

Iteration 3

$f(x) = 5\pi x - x^2$, $[6.25, 8.75]$

$$h = \frac{8.75 - 6.25}{4} = 0.625$$

~~0.625~~

x_5	6.25	59.11
x_7	6.875	60.73
x_3	7.5	61.56
x_8	8.125	~~60.88~~ 61.61
x_6	8.75	~~[illegible]~~ 60.88

Centre is at x_8. SI $[7.5, 8.75]$

$|f(x_8) - f(x_3)| =$ ~~$|61.61 - 60.|$~~ ~~$|61.56 - 60.88|$~~

$|61.61 - 61.56| = 0.05 < G$

Ans max value is 61.61 at x_8 //

MQP 7 a)

Use Steepest Descent Method for $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$ starting from $X_1 = (0,0)$

Ans $\nabla f = \begin{pmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \end{pmatrix} = \begin{pmatrix} 1+4x_1+2x_2 \\ -1+2x_1+2x_2 \end{pmatrix}$

Hessian Matrix $H = \begin{pmatrix} \partial^2 f/\partial x_1^2 & \partial^2 f/\partial x_1 \partial x_2 \\ \partial^2 f/\partial x_2 \partial x_1 & \partial^2 f/\partial x_2^2 \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ 2 & 2 \end{pmatrix}$

Iteration 1 Step 1

Find S_1 at $X_1 = (0,0)$

$S_1 = -\nabla f(X_1) = -\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

Step 2 Compute λ_1 at X_1

$\lambda_1 = \dfrac{S_1^T S_1}{S_1^T H S_1} = \dfrac{\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}}{\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix}} = 1$

New point $X_2 = X_1 + \lambda_1 S_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + 1\begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

Step 3 Check the optimum

$\nabla f(X_2) = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ not optimum

Iteration 2 Step 1 Find S_2 at $X_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

$S_2 = -\nabla f(X_2) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

Step 2 Compute λ_2 at X_2

$\lambda_2 = \dfrac{S_2^T S_2}{S_2^T H S_2} = \dfrac{\begin{pmatrix} 1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}}{\begin{pmatrix} 1 & 1 \end{pmatrix}\begin{pmatrix} 4 & 2 \\ 2 & 2 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}} = \dfrac{1}{5}$

Hence the new point

$\therefore x_3 = x_2 + \lambda_2 S_2 = \begin{pmatrix}-1\\1\end{pmatrix} + \frac{1}{5}\begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}-0.8\\1.2\end{pmatrix}$

Step 3 Check the optimum

$\nabla f(x_3) = \begin{pmatrix}+0.2\\-0.2\end{pmatrix} \neq \begin{pmatrix}0\\0\end{pmatrix}$ not an optimum

x_3 is not optimum

Iteration 3 Step 1 $x_3 = \begin{pmatrix}-0.8\\1.2\end{pmatrix}$

$S_3 = -\nabla f(x_3) = -\begin{pmatrix}0.2\\-0.2\end{pmatrix} = \begin{pmatrix}-0.2\\0.2\end{pmatrix}$

Step 2 $\lambda_3 = \dfrac{S_3^T S_3}{S_3^T H S_3} = \dfrac{\begin{pmatrix}-0.2 & 0.2\end{pmatrix}\begin{pmatrix}-0.2\\0.2\end{pmatrix}}{\begin{pmatrix}-0.2 & 0.2\end{pmatrix}\begin{pmatrix}4 & 2\\2 & 2\end{pmatrix}\begin{pmatrix}-0.2\\0.2\end{pmatrix}}$

$= 1$

Hence the new point is

$x_4 = x_3 + \lambda_3 S_3 = \begin{pmatrix}-0.8\\1.2\end{pmatrix} + 1\begin{pmatrix}-0.2\\0.2\end{pmatrix}$

$= \begin{pmatrix}-1\\1.4\end{pmatrix}$

Step 3 Check the optimum

$\nabla f(x_4) = \begin{pmatrix}-0.2\\-0.2\end{pmatrix} \neq \begin{pmatrix}0\\0\end{pmatrix}$

x_4 is not an optimum

Iteration 4 Step 1 $S_4 = -\nabla f(x_4) = \begin{pmatrix}0.2\\0.2\end{pmatrix}$

Step 2 $\lambda_4 = \dfrac{S_4^T S_4}{S_4^T H S_4} = \dfrac{\begin{pmatrix}0.2 & 0.2\end{pmatrix}\begin{pmatrix}0.2\\0.2\end{pmatrix}}{\begin{pmatrix}0.2 & 0.2\end{pmatrix}\begin{pmatrix}4 & 2\\2 & 2\end{pmatrix}\begin{pmatrix}0.2\\0.2\end{pmatrix}}$

$= 1/5$

Hence the new point is $X_5 = X_4 + \lambda_4 S_4$

$$= \begin{pmatrix} -1 \\ 1.4 \end{pmatrix} + \frac{1}{5}\begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix} = \begin{pmatrix} -0.96 \\ 1.44 \end{pmatrix}$$

Step 3 $\nabla f(X_5) = \begin{pmatrix} 1 + 4(-0.96) + 2(1.44) \\ -1 + 2(-0.96) + 2(1.44) \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

X_5 is an optimum

Q8 a) Use Newton Raphson method to find the smallest and the second smallest positive roots of the eqn $\tan x = 4x$ correct to 4 decimal places.

Ans Draw the curves $y = \tan x$ $y = 4x$.
The roots of the eqn are the x co-ordinates of the places where the 2 curves meet.
The 2 curves meet at $x = 0$, then at a point with x just shy of $\frac{\pi}{2}$ and then again at a point with x with x just shy of $\frac{3\pi}{2}$

We find that the root is near $\frac{\pi}{2}$.

$$f(x) = \tan x - 4x \qquad f'(x) = \sec^2 x - 4$$

Newton Method of Recurrence

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$x_{n+1} = x_n - \frac{\tan x_n - 4x_n}{\sec^2 x_n - 4} = x_n - \frac{\tan x_n - 4x_n}{\tan^2 x_n - 3}$$

First root near $x_0 = 1.4$

$$x_1 = x_0 - \frac{\tan x_0 - 4x_0}{\tan^2 x_0 - 3}$$ at $x_0 = 1.4$ is 1.3935

$$x_2 = x_1 - \frac{\tan x_1 - 4x_1}{\tan^2 x_1 - 3}$$ at $x_1 = 1.3935$ is 1.3932

$$x_3 = x_2 - \frac{\tan x_2 - 4x_2}{\tan^2 x_2 - 3}$$ at $x_2 = 1.3932$ is 1.3932

For the second root near $x_0 = 4.66$

$$x_1 = x_0 - \frac{\tan x_0 - 4x_0}{\tan^2 x_0 - 3} \quad \text{at } x_0 = 4.66 \text{ is } 4.6588$$

$$x_2 = x_1 - \frac{\tan x_1 - 4x_1}{\tan^2 x_1 - 3} \quad \text{at } x_1 = 4.6588 \text{ is } 4.6588$$

$$x_3 = x_2 - \frac{\tan x_2 - 4x_2}{\tan^2 x_2 - 3} \quad \text{at } x_2 = 4.6588 \text{ is } 4.6588$$

Root is 4.6588

collects partial derivatives. For example, if we compute the gradient of an $m \times n$ matrix $\boldsymbol{A}$ with respect to a $p \times q$ matrix $\boldsymbol{B}$, the resulting Jacobian would be $(m\times n)\times(p\times q)$, i.e., a four-dimensional tensor $\boldsymbol{J}$, whose entries are given as $J_{ijkl} = \partial A_{ij}/\partial B_{kl}$.

Since matrices represent linear mappings, we can exploit the fact that there is a vector-space isomorphism (linear, invertible mapping) between the space $\mathbb{R}^{m\times n}$ of $m \times n$ matrices and the space $\mathbb{R}^{mn}$ of mn vectors. Therefore, we can re-shape our matrices into vectors of lengths mn and pq, respectively. The gradient using these mn vectors results in a Jacobian of size $mn \times pq$. Figure 5.7 visualizes both approaches. In practical applications, it is often desirable to re-shape the matrix into a vector and continue working with this Jacobian matrix: The chain rule (5.48) boils down to simple matrix multiplication, whereas in the case of a Jacobian tensor, we will need to pay more attention to what dimensions we need to sum out.

Matrices can be transformed into vectors by stacking the columns of the matrix (“flattening”).

Example 5.12 (Gradient of Vectors with Respect to Matrices)
Let us consider the following example, where

$$\boldsymbol{f} = \boldsymbol{A}\boldsymbol{x}\,, \quad \boldsymbol{f} \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N \tag{5.85}$$

and where we seek the gradient $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{A}$. Let us start again by determining the dimension of the gradient as

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{A}} \in \mathbb{R}^{M\times(M\times N)}\,. \tag{5.86}$$

By definition, the gradient is the collection of the partial derivatives:

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{A}} = \begin{bmatrix} \frac{\partial f_1}{\partial \boldsymbol{A}} \\ \vdots \\ \frac{\partial f_M}{\partial \boldsymbol{A}} \end{bmatrix}, \quad \frac{\partial f_i}{\partial \boldsymbol{A}} \in \mathbb{R}^{1\times(M\times N)}\,. \tag{5.87}$$

To compute the partial derivatives, it will be helpful to explicitly write out the matrix vector multiplication:

$$f_i = \sum_{j=1}^{N} A_{ij}x_j, \quad i = 1, \dots, M\,, \tag{5.88}$$

and the partial derivatives are then given as

$$\frac{\partial f_i}{\partial A_{iq}} = x_q\,. \tag{5.89}$$

This allows us to compute the partial derivatives of f_i with respect to a row of $\boldsymbol{A}$, which is given as

$$\frac{\partial f_i}{\partial A_{i,:}} = \boldsymbol{x}^\top \in \mathbb{R}^{1\times 1\times N}\,, \tag{5.90}$$

$$\frac{\partial f_i}{\partial A_{k\neq i,:}} = \mathbf{0}^\top \in \mathbb{R}^{1\times 1\times N} \tag{5.91}$$

where we have to pay attention to the correct dimensionality. Since f_i maps onto $\mathbb{R}$ and each row of $\boldsymbol{A}$ is of size $1 \times N$, we obtain a $1 \times 1 \times N$-sized tensor as the partial derivative of f_i with respect to a row of $\boldsymbol{A}$.

We stack the partial derivatives (5.91) and get the desired gradient in (5.87) via

$$\frac{\partial f_i}{\partial \boldsymbol{A}} = \begin{bmatrix} \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \\ \boldsymbol{x}^\top \\ \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \end{bmatrix} \in \mathbb{R}^{1\times(M\times N)} . \tag{5.92}$$

Example 5.13 (Gradient of Matrices with Respect to Matrices)
Consider a matrix $\boldsymbol{R} \in \mathbb{R}^{M\times N}$ and $\boldsymbol{f} : \mathbb{R}^{M\times N} \to \mathbb{R}^{N\times N}$ with

$$\boldsymbol{f}(\boldsymbol{R}) = \boldsymbol{R}^\top \boldsymbol{R} =: \boldsymbol{K} \in \mathbb{R}^{N\times N} , \tag{5.93}$$

where we seek the gradient $\mathrm{d}\boldsymbol{K}/\mathrm{d}\boldsymbol{R}$.

To solve this hard problem, let us first write down what we already know: The gradient has the dimensions

$$\frac{\mathrm{d}\boldsymbol{K}}{\mathrm{d}\boldsymbol{R}} \in \mathbb{R}^{(N\times N)\times(M\times N)} , \tag{5.94}$$

which is a tensor. Moreover,

$$\frac{\mathrm{d}K_{pq}}{\mathrm{d}\boldsymbol{R}} \in \mathbb{R}^{1\times M\times N} \tag{5.95}$$

for $p, q = 1, \ldots, N$, where K_{pq} is the (p, q)th entry of $\boldsymbol{K} = \boldsymbol{f}(\boldsymbol{R})$. Denoting the ith column of $\boldsymbol{R}$ by $\boldsymbol{r}_i$, every entry of $\boldsymbol{K}$ is given by the dot product of two columns of $\boldsymbol{R}$, i.e.,

$$K_{pq} = \boldsymbol{r}_p^\top \boldsymbol{r}_q = \sum_{m=1}^{M} R_{mp} R_{mq} . \tag{5.96}$$

When we now compute the partial derivative $\frac{\partial K_{pq}}{\partial R_{ij}}$ we obtain

$$\frac{\partial K_{pq}}{\partial R_{ij}} = \sum_{m=1}^{M} \frac{\partial}{\partial R_{ij}} R_{mp} R_{mq} = \partial_{pqij} , \tag{5.97}$$

$$\partial_{pqij} = \begin{cases} R_{iq} & \text{if } j = p,\ p \neq q \\ R_{ip} & \text{if } j = q,\ p \neq q \\ 2R_{iq} & \text{if } j = p,\ p = q \\ 0 & \text{otherwise} \end{cases} . \tag{5.98}$$

From (5.94), we know that the desired gradient has the dimension $(N \times N) \times (M \times N)$, and every single entry of this tensor is given by ∂_{pqij} in (5.98), where $p, q, j = 1, \ldots, N$ and $i = 1, \ldots, M$.

5.5 Useful Identities for Computing Gradients

In the following, we list some useful gradients that are frequently required in a machine learning context (Petersen and Pedersen, 2012). Here, we use $\mathrm{tr}(\cdot)$ as the trace (see Definition 4.4), $\det(\cdot)$ as the determinant (see Section 4.1) and $\boldsymbol{f}(\boldsymbol{X})^{-1}$ as the inverse of $\boldsymbol{f}(\boldsymbol{X})$, assuming it exists.

$$\frac{\partial}{\partial \boldsymbol{X}} \boldsymbol{f}(\boldsymbol{X})^\top = \left(\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right)^\top \tag{5.99}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \mathrm{tr}(\boldsymbol{f}(\boldsymbol{X})) = \mathrm{tr}\left(\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right) \tag{5.100}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \det(\boldsymbol{f}(\boldsymbol{X})) = \det(\boldsymbol{f}(\boldsymbol{X}))\mathrm{tr}\left(\boldsymbol{f}(\boldsymbol{X})^{-1}\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right) \tag{5.101}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \boldsymbol{f}(\boldsymbol{X})^{-1} = -\boldsymbol{f}(\boldsymbol{X})^{-1}\frac{\partial \boldsymbol{f}(\boldsymbol{X})}{\partial \boldsymbol{X}}\boldsymbol{f}(\boldsymbol{X})^{-1} \tag{5.102}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{X}^{-1}\boldsymbol{b}}{\partial \boldsymbol{X}} = -(\boldsymbol{X}^{-1})^\top \boldsymbol{a}\boldsymbol{b}^\top (\boldsymbol{X}^{-1})^\top \tag{5.103}$$

$$\frac{\partial \boldsymbol{x}^\top \boldsymbol{a}}{\partial \boldsymbol{x}} = \boldsymbol{a}^\top \tag{5.104}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{a}^\top \tag{5.105}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{X}\boldsymbol{b}}{\partial \boldsymbol{X}} = \boldsymbol{a}\boldsymbol{b}^\top \tag{5.106}$$

$$\frac{\partial \boldsymbol{x}^\top \boldsymbol{B}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{x}^\top(\boldsymbol{B} + \boldsymbol{B}^\top) \tag{5.107}$$

$$\frac{\partial}{\partial \boldsymbol{s}}(\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s})^\top \boldsymbol{W}(\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}) = -2(\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s})^\top \boldsymbol{W}\boldsymbol{A} \quad \text{for symmetric } \boldsymbol{W} \tag{5.108}$$

Remark. In this book, we only cover traces and transposes of matrices. However, we have seen that derivatives can be higher-dimensional tensors, in which case the usual trace and transpose are not defined. In these cases, the trace of a $D \times D \times E \times F$ tensor would be an $E \times F$-dimensional matrix. This is a special case of a tensor contraction. Similarly, when we

```
  x   y
  |   |
  |   +------+
  |          |
  |          v
  |        [sin]
  |          |
  |          v
[log]     [-sin]
  |          |
  |          v
  +----> [xy]
       |   |
       v   v
     [sum]
       |
       v
     Output
```

To draw the computational graph for the function f(x,y)=log(x)+xy−sin(y), we need to break down the function into its individual operations and represent them as nodes in the graph. Here's how you can structure it:

1. **Input Nodes**:
   - x
   - y
2. **Intermediate Computations**:
   - log(x) (logarithm node)
   - xy (multiplication node)
   - sin(y) (sine node)
3. **Output Computation**:
   - Sum the results of the nodes from the intermediate computations: log(x)+xy−sin(y)

In this graph:

- x and y are input variables.
- The node [log] takes x as input and computes log(x).
- The node [xy] computes the product of x and y.
- The node [sin] computes sin(y), and the [-sin] node negates it.
- The [sum] node adds the results of log],[xy], and [−sin] to produce the final output.

- **Q5a) Difference between local optima and global optima**
- **Definition**:
  - **Local Optima**: A point where a function's value is better (higher for maxima, lower for minima) than the values of all nearby points, but not necessarily the best overall.
  - **Global Optima**: A point where a function's value is the best overall across the entire domain of the function.
- **Scope**:
  - **Local Optima**: Limited to a neighborhood or small region of the function.
  - **Global Optima**: Considers the entire range or domain of the function.
- **Objective**:
  - **Local Optima**: Indicates a solution that is optimal within a limited scope, but there may be better solutions elsewhere in the domain.
  - **Global Optima**: Represents the best possible solution across the entire domain, with no better solutions available.
- **Complexity in Finding**:
  - **Local Optima**: Easier to find, as optimization algorithms often converge to local optima based on the starting point and the algorithm used.
  - **Global Optima**: Harder to find, especially in complex or non-convex functions, as it requires exploring the entire domain to ensure no better solutions exist.
- **Significance in Optimization**:
  - **Local Optima**: May be satisfactory for certain applications, especially if global optimization is computationally expensive or unnecessary.
  - **Global Optima**: Ideal for applications where the best possible solution is required, and no compromises can be made.
- **Examples**:
  - In a landscape with multiple hills and valleys, the tops of individual hills are local maxima, while the highest hilltop is the global maximum.

Q5c Explain the algorithm of sequential search

What is Linear Search Algorithm?

Linear search is a method for searching for an element in a collection of elements. In linear search, each element of the collection is visited one by one in a sequential fashion to find the desired element. Linear search is also known as **sequential search**.

Algorithm for Linear Search Algorithm:

The algorithm for linear search can be broken down into the following steps:

- **Start:** Begin at the first element of the collection of elements.
- **Compare:** Compare the current element with the desired element.
- **Found:** If the current element is equal to the desired element, return true or index to the current element.

- **Move:** Otherwise, move to the next element in the collection.
- **Repeat:** Repeat steps 2-4 until we have reached the end of collection.
- **Not found:** If the end of the collection is reached without finding the desired element, return that the desired element is not in the array.

How Does Linear Search Algorithm Work?

In Linear Search Algorithm,

- Every element is considered as a potential match for the key and checked for the same.
- If any element is found equal to the key, the search is successful and the index of that element is returned.
- If no element is found equal to the key, the search yields “No match found

Q6b) Write the algorithm of Fibonacci Search Algorithm

The Fibonacci Search Algorithm makes use of the Fibonacci Series to diminish the range of an array on which the searching is set to be performed. With every iteration, the search range decreases making it easier to locate the element in the array. The detailed procedure of the searching is seen below −

Step 1 − As the first step, find the immediate Fibonacci number that is greater than or equal to the size of the input array. Then, also hold the two preceding numbers of the selected Fibonacci number, that is, we hold Fm, Fm-1, Fm-2 numbers from the Fibonacci Series.

Step 2 − Initialize the offset value as -1, as we are considering the entire array as the searching range in the beginning.

Step 3 − Until Fm-2 is greater than 0, we perform the following steps −

- Compare the key element to be found with the element at index **[min(offset+F_{m-2},n-1)]**. If a match is found, return the index.
- If the key element is found to be lesser value than this element, we reduce the range of the input from 0 to the index of this element. The Fibonacci numbers are also updated with $F_m = F_{m-2}$.
- But if the key element is greater than the element at this index, we remove the elements before this element from the search range. The Fibonacci numbers are updated as $F_m = F_{m-1}$. The *offset* value is set to the index of this element.

Step 4 − As there are two 1s in the Fibonacci series, there arises a case where your two preceding numbers will become 1. So if F_{m-1} becomes 1, there is only one element left in the array to be searched. We compare the key element with that element and return the 1st index. Otherwise, the algorithm returns an unsuccessful search.

```
Begin Fibonacci Search
   n <- size of the input array
   offset = -1
   Fm2 := 0
   Fm1 := 1
   Fm := Fm2 + Fm1
   while Fm < n do:
      Fm2 = Fm1
      Fm1 = Fm
      Fm = Fm2 + Fm1
   done
   while fm > 1 do:
      i := minimum of (offset + fm2, n − 1)
      if (A[i] < x) then:
         Fm := Fm1
         Fm1 := Fm2
         Fm2 := Fm - Fm1
         offset = i
      end
      else if (A[i] > x) then:
         Fm = Fm2
         Fm1 = Fm1 - Fm2
         Fm2 = Fm - Fm1
      end
      else
         return i;
      end
   done
   if (Fm1 and Array[offset + 1] == x) then:
      return offset + 1
   end
   return invalid location;
end
```

Analysis

The Fibonacci Search algorithm takes logarithmic time complexity to search for an element. Since it is based on a divide on a conquer approach and is similar to idea of binary search, the time taken by this algorithm to be executed under the worst case consequences is **O(log n).**

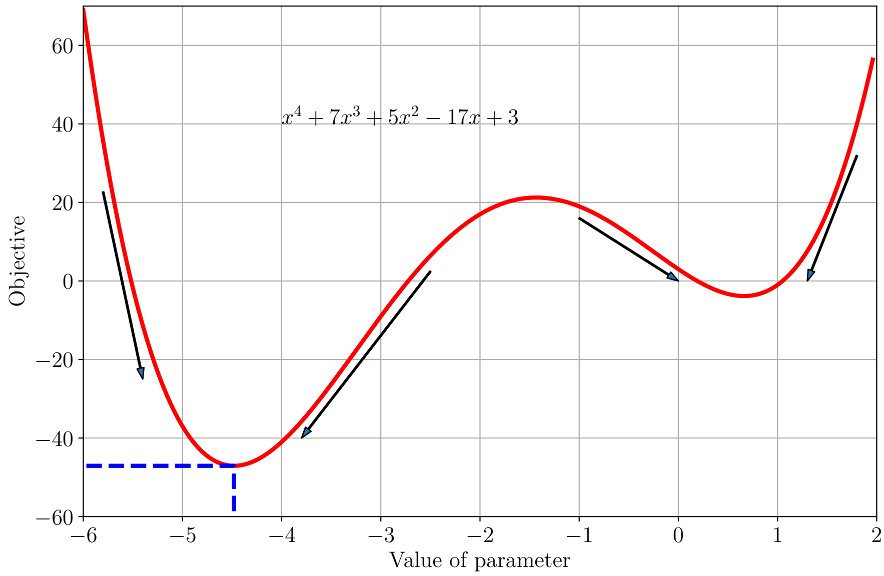


Figure 7.2 Example objective function. Negative gradients are indicated by arrows, and the global minimum is indicated by the dashed blue line.

right, but not how far (this is called the step-size). Furthermore, if we had started at the right side (e.g., $x_0 = 0$) the negative gradient would have led us to the wrong minimum. Figure 7.2 illustrates the fact that for $x > -1$, the negative gradient points toward the minimum on the right of the figure, which has a larger objective value.

According to the Abel–Ruffini theorem, there is in general no algebraic solution for polynomials of degree 5 or more (Abel, 1826).

In Section 7.3, we will learn about a class of functions, called convex functions, that do not exhibit this tricky dependency on the starting point of the optimization algorithm. For convex functions, all local minimums are global minimum. It turns out that many machine learning objective functions are designed such that they are convex, and we will see an example in Chapter 12.

For convex functions all local minima are global minimum.

The discussion in this chapter so far was about a one-dimensional function, where we are able to visualize the ideas of gradients, descent directions, and optimal values. In the rest of this chapter we develop the same ideas in high dimensions. Unfortunately, we can only visualize the concepts in one dimension, but some concepts do not generalize directly to higher dimensions, therefore some care needs to be taken when reading.

7.1 Optimization Using Gradient Descent

We now consider the problem of solving for the minimum of a real-valued function

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) , \tag{7.4}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is an objective function that captures the machine learning problem at hand. We assume that our function f is differentiable, and we are unable to analytically find a solution in closed form.

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point. Recall from Section 5.1 that the gradient points in the direction of the steepest ascent. Another useful intuition is to consider the set of lines where the function is at a certain value ($f(\boldsymbol{x}) = c$ for some value $c \in \mathbb{R}$), which are known as the contour lines. The gradient points in a direction that is orthogonal to the contour lines of the function we wish to optimize.

We use the convention of row vectors for gradients.

Let us consider multivariate functions. Imagine a surface (described by the function $f(\boldsymbol{x})$) with a ball starting at a particular location $\boldsymbol{x}_0$. When the ball is released, it will move downhill in the direction of steepest descent. Gradient descent exploits the fact that $f(\boldsymbol{x}_0)$ decreases fastest if one moves from $\boldsymbol{x}_0$ in the direction of the negative gradient $-((\nabla f)(\boldsymbol{x}_0))^\top$ of f at $\boldsymbol{x}_0$. We assume in this book that the functions are differentiable, and refer the reader to more general settings in Section 7.4. Then, if

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \gamma((\nabla f)(\boldsymbol{x}_0))^\top \tag{7.5}$$

for a small *step-size* $\gamma \geqslant 0$, then $f(\boldsymbol{x}_1) \leqslant f(\boldsymbol{x}_0)$. Note that we use the transpose for the gradient since otherwise the dimensions will not work out.

This observation allows us to define a simple gradient descent algorithm: If we want to find a local optimum $f(\boldsymbol{x}_*)$ of a function $f : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{x} \mapsto f(\boldsymbol{x})$, we start with an initial guess $\boldsymbol{x}_0$ of the parameters we wish to optimize and then iterate according to

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla f)(\boldsymbol{x}_i))^\top . \tag{7.6}$$

For suitable step-size γ_i, the sequence $f(\boldsymbol{x}_0) \geqslant f(\boldsymbol{x}_1) \geqslant \ldots$ converges to a local minimum.

Example 7.1

Consider a quadratic function in two dimensions

$$f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^\top \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.7}$$

with gradient

$$\nabla f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^\top . \tag{7.8}$$

Starting at the initial location $\boldsymbol{x}_0 = [-3, -1]^\top$, we iteratively apply (7.6) to obtain a sequence of estimates that converge to the minimum value

Working Procedure to Fit a Linear Regression Line to Data Using the Gradient Descent Method

(1) The following steps are used to fit a linear regression line $\hat{y} = a + bx$ for the given data:

Step 1: Define the Model and Cost Function

The linear regression model you want to fit is given by:

$$\hat{y} = a + bx$$

where ŷ is the predicted value, *a* and *b* are the parameters (weights) of the model and *x* is the input feature.

The cost function (error function) for linear regression is the Mean Squared Error (MSE):

$$MSE = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y})^2$$

$$MSE = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - (a + bx_i))^2$$

Where *n* is the number of data points.

Step 2: Initialize Weights and Hyperparameters

Initialize the weights *a* and *b* to some arbitrary values. Start with $a = 0$ and $b = 0$. Also need to set hyperparameters:

- Learning rate (α): A small positive value that controls the step size in each iteration.
- Number of iterations: The number of times we update the weights.

Professor, Department of Artificial Intelligence & Data Science,
Don Bosco Institute of Technology, Bangalore.

Step 3: Gradient Descent

For each iteration, calculate the gradients of the cost function with respect to the weights (*a* and *b*) and update the weights accordingly.

The gradients are given by:

$$\Delta a = -\frac{2}{n}\sum_{i=1}^{n}\left(y_i - (a + bx_i\,)\right)$$

$$\Delta b = -\frac{2}{n}\sum_{i=1}^{n}x_i \times \left(y_i - (a + bx_i\,)\right)$$

Update the weights using the gradients:

$$a_{new} = a - \alpha \times \Delta a$$

$$b_{new} = b - \alpha \times \Delta b$$

Repeat this process for the specified number of iterations.

Step 4: Predict

After training, use the final values of *a* and *b* to make predictions:

$$\hat{y} = a + bx$$

Step 5: Evaluate and visualize

Evaluate the quality of the linear regression model by calculating the final MSE on your training data:

$$MSE = \frac{1}{n} \times \sum_{i=1}^{n}(y_i - \hat{y}\,)^2$$

Also, visualize the linear regression line by plotting it alongside the data points.

Step 6: Iterate as Needed

We need to adjust the learning rate and the number of iterations to find the best-fitting line. If the cost is not converging or fluctuating, you may need to modify the hyperparameters.

This process allows you to iteratively update the weights to minimize the cost function, resulting in a linear regression line that best fits the given data.

Examples

1. We have recorded the weekly average price of a stock over 6 consecutive days. Y shows the weekly average price of the stock and x shows the number of the days. Try to fit the best possible function ' f ' to establish the relationship between the number of the day and conversion rate. (Applying Gradient descent) where f(x) = y = a + b * x.

X	1	2	3	4	5	6
Y	10	14	18	22	25	33

The initial values of a & b are a = 4.9 & b = 4.401. The learning rate is mentioned as .05. The error rate of a & b should be less than .01. Plot the predicted and actual data in a graph.

Solution:

Given data:

X = x_i = 1, 2, 3, 4, 5, 6

Y = y_i = 10, 14, 18, 22, 25, 33

n = 6

Initialization: a = 4. 9 and b = 4.401

Learning rate α = 0.05

Maximum allowable error for a and b = 0.01

The goal is to minimize the MSE (mean squared error) defined as

Professor, Department of Artificial Intelligence & Data Science,
Don Bosco Institute of Technology, Bangalore.

Q8b)Write the differences between SGD and mini batch gradient descent methods.

Stochastic gradient descent (SGD) and mini-batch gradient descent are both variants of the gradient descent algorithm, which is an optimization algorithm used in machine learning. The main difference between them is the amount of training data used in each iteration:

- Stochastic gradient descent (SGD)

  Uses a single example or a small subset of examples in each iteration. SGD is faster than mini-batch gradient descent (MGD) and batch gradient descent (BGD) because it doesn't need to wait for the entire dataset to calculate itself. SGD can be used for larger datasets and is useful in machine learning, geophysics, and least mean squares (LMS). However, due to its random nature, SGD may not provide the exact solution, but rather the best approximate solution.

- Mini-batch gradient descent (MGD)

  Uses a fixed number of training examples, called a mini-batch, that is less than the entire dataset. MGD helps to combine the advantages of both SGD and batch gradient descent.

Batch Gradient Descent	Stochastic Gradient Descent
Computes gradient using the whole Training sample	Computes gradient using a single Training sample
Slow and computationally expensive algorithm	Faster and less computationally expensive than Batch GD
Not suggested for huge training samples.	Can be used for large training samples.
Deterministic in nature.	Stochastic in nature.
Gives optimal solution given sufficient time to converge.	Gives good solution but not optimal.

Batch Gradient Descent	Stochastic Gradient Descent
No random shuffling of points are required.	The data sample should be in a random order, and this is why we want to shuffle the training set for every epoch.
Can’t escape shallow local minima easily.	SGD can escape shallow local minima more easily.
Convergence is slow.	Reaches the convergence much faster.
It updates the model parameters only after processing the entire training set.	It updates the parameters after each individual data point.
The learning rate is fixed and cannot be changed during training.	The learning rate can be adjusted dynamically.
It typically converges to the global minimum for convex loss functions.	It may converge to a local minimum or saddle point.
It may suffer from overfitting if the model is too complex for the dataset.	It can help reduce overfitting by updating the model parameters more frequently.

Q9b What is the difference between convex optimization and non convex optimization?

Convex optimization and non-convex optimization are both optimization problems, but they differ in the number of optimal solutions they can have:

- Convex optimization

In convex optimization, there can only be one globally optimal solution, or it may be possible to prove that there is no feasible solution. Convex optimization is easier and more reliable because convex functions have a unique global minimum. Convex problems can also be solved efficiently, even when they are very large. Examples of convex optimization problems include multi-period processor speed scheduling, minimum time optimal control, and grasp force optimization.

- Non-convex optimization

  In non-convex optimization, the objective or some of the constraints are non-convex, which can lead to multiple feasible regions and multiple locally optimal points within each region. This can make optimization more challenging. Non-convex optimization can still be a good choice if the optimization scheme doesn't get stuck in a local minimum. It can also be used to implement more accurate state dynamics. However, even simple-looking non-convex optimization problems with only ten variables can be very challenging, and problems with hundreds of variables can be intractable.

Experiment No.   Page No.:

Stochastic Gradient Descent (SGD)

Computing the gradient can be very time consuming. Approximating the gradient is useful as long as it points in roughly the same direction as the true gradient.

SGD is a stochastic approximation of ~~the~~ gradient descent method for minimizing an objective function that is written as a sum of differentiable functions.

In this method, we start with a noisy approximation though we do not know the gradient precisely.

By constraining the probability distribution of the approximate gradients, we can guarantee that SGD will converge.

Given $n = 1, 2, \ldots, N$ data points, we consider the sum of the losses L_n incurred by each example n

Thus, we have

$$L(\theta) = \sum_{\theta=1}^{N} L_n(\theta), \text{ where}$$

θ is the vector of parameters of intrest

Aim is to find θ that minimizes L.

For example,

$$L(\theta) = -\sum \log P(y_n | x_n, \theta)$$

where $x_n \in R^D$ are training inputs, y_n are training targets

described loss function for, Stochastic gradient descent

Experiment No. Page No.:

Standard gradient descent is a "batch" optimization method, in which, optimization is performed using the full training set. by updating θ according to

$$\theta_{i+1} = \theta_i - \gamma_i (\nabla L(\theta_i))^T$$

$$= \theta_i - \gamma_i \sum_{n=1}^{N} (\nabla L_n(\theta_i))^T$$

If the sum is taken over a small set of L_n i.e a subset of L_n, then the method is known as mini batch gradient descent.

10a)Stochastic Gradient Descent with Momentum

The first of the four algorithms I would like to introduce is called “Stochastic Gradient Descent with Momentum”:

SGD

$$\theta_j \leftarrow \theta_j - \epsilon \nabla_{\theta_j} \mathcal{L}(\theta)$$

SGD mit Impuls

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$

$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

GL. 2 Stochastic GD (left), SGD with momentum (right).

On the left side in GL. 2 is the formula for the weight updates according to the regular stochastic gradient descent (SGD for short). The equation on the right represents the rule for the updates of the weights according to the SGD with momentum. **Momentum appears here as an additional term** , which is added to the regular update rule.

Intuitively speaking, by adding this impulse term, we let our **gradient build up some sort of velocity** V during training. The velocity is the running sum of the gradients weighted by ρ.

The parameter ρ can be thought of as friction that “slows” the velocity down a bit. In general, velocity can be seen to increase with time. By using the momentum term, **saddle points and local minima become less dangerous** for the gradient. This is because the step size toward the global minimum now depends not only on the slope of the loss function at the current point, but **also on the velocity** that has built up over time.

For a physical representation of stochastic gradient descent with momentum, imagine a ball rolling down a hill, increasing in velocity with time. If this ball encounters an obstacle along the way, such as a hole or flat ground with no slope, its built-up velocity v would give the ball enough force to roll over this obstacle. In this case, the **flat ground represents a saddle point** and the **hole represents a local minima** of a loss function.

Both algorithms try to reach the global minimum of the loss function, which is in a 3D space. Momentum term results in the individual gradients having less variance and thus less zig-zagging.

10a)ii)ADAM

We take the best of Adagrad and RMS prop and combine these ideas into a single algorithm called as ADAM.

The main part of this optimization algorithm consists of the following three equations. These equations may seem complicated at first glance, but if you look closely, you will see some similarities with the last three optimization algorithms.

$$m_0 = 0, v_0 = 0$$

$$\boxed{m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1)\nabla_\theta \mathcal{L}(\theta)}$$ Impuls

$$\boxed{v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2)\nabla_\theta \mathcal{L}(\theta)^2}$$ RMS Prop

$$\boxed{\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}}$$ RMS Prop + Impuls

The first expression looks a bit like SGD with momentum. In this case, the term m_t would be the velocity and the term β_1 would be the friction term. In the case of ADAM, we refer to m_t as the "first momentum." On the other hand, β_1 is just a hyperparameter. However, the difference with SGD with momentum is the factor $1-\beta_1$ multiplied by the current gradient.

The second expression **can be considered as RMSProp**, where we keep the running sum of squared gradients. Also in this case, there is the factor $1-\beta_2$, which is multiplied by the squared gradient.

The term v_t in the equation is called the “second momentum” and is also just a hyperparameter. The final update equation can be viewed as **a combination of RMSProp and SGD with momentum**.

Disadvantages

At the very first time step t=0, the first and second pulse terms m_0 and v_0 are set to zero. After the first update of the second momentum v_1, this term is still very close to zero. When we update the weight parameters in the last expression in GL. 5, we divide by a very small second momentum term v_1. This leads to a very large first update step.

To address the problem of large update steps happening at the beginning of training, ADAM includes a correction clause:

$$\boxed{m_{t+1} \leftarrow \beta_1 m_t + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta)} \quad \text{Impuls}$$

$$\boxed{v_{t+1} \leftarrow \beta_2 v_t + (1-\beta_2)\nabla_\theta \mathcal{L}(\theta)^2} \quad \text{RMS Prop}$$

$$\boxed{\begin{aligned} \hat{m}_{t+1} &\leftarrow \frac{m_{t+1}}{1-\beta_1^t} \\ \hat{v}_{t+1} &\leftarrow \frac{v_{t+1}}{1-\beta_2^t} \end{aligned}} \quad \text{Bias Korrektur}$$

$$\boxed{\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}} \quad \text{RMS Prop + Impuls}$$

After the initial update of the first and second pulses, we **make an unbiased estimate of these pulses** by considering the current time step. With the so-called bias correction, we obtain the corrected first and second impulses respectively.

These correction cause the values of the first and second impulse to be higher at the beginning of the training than without this correction. As a result, the first update step of the neural network weight parameters does not become too large. Thus, the training is not already messed up at the very beginning.

With the additional bias corrections, we obtain the complete form of the ADAM optimizer.

9a)AdaGrad optimization strategy

Another optimization strategy I would like to introduce is called AdaGrad. The idea behind AdaGrad is that you keep a running sum of squared gradients during optimization. In this case, we don't have a momentum term, but an expression , which is **the sum of squared gradients** up to the time .

SGD mit Impuls

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$

$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

AdaGrad

$$g_0 = 0$$

$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

When we optimize a weights θj , we divide the current gradient $\nabla_j L$ by the root of the term g $_{t+1}$. To understand the intuition behind AdaGrad, please imagine a loss function in a two-dimensional space. In this space, the gradient of the loss function **increases very weakly in one direction** and **very strongly in the other direction**. If we now sum up the gradients along the axis in which the gradients increase weakly, the squared sum of these gradients becomes even smaller.

If during the update step we divide the current gradient $\nabla_j L$ by a very small sum of the squared gradients g_{t+1} , the quotient becomes very high. For the other axis, along which the gradients increase sharply, exactly the opposite is true. This means that we speed up the updating process **along the axis with weak gradients** by increasing these gradients along this axis. On the other hand, we slow down the updates of the weights **along the axis with large gradients**.

Disadvantages: there is a problem with this optimization algorithm.

If the training takes too long. Over time, this term the sum of squared gradients would **grow larger.** When the current gradient is divided by this large number, the update step for the weights becomes very small. It is as if we were using **a very low learning rate**, which becomes even lower the longer the training takes. In the worst case, we would get stuck at AdaGrad and the training would go on forever.

9a) &10c)RMSProp

There is a slight modification of AdaGrad called “RMSProp”. This modification is intended to solve the previously described problem that can occur with AdaGrad. In RMSProp, the running sum of squared gradients g_{t+1} is maintained. However, instead of allowing this sum to increase continuously over the training period, we allow the sum to decrease.

AdaGrad	RMS Prop
$g_0 = 0$	$g_0 = 0, \alpha \simeq 0.9$
$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$	$g_{t+1} \leftarrow \alpha \cdot g_t + (1-\alpha)\nabla_\theta \mathcal{L}(\theta)^2$
$\theta_j \leftarrow \theta_j - \epsilon \dfrac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$	$\theta_j \leftarrow \theta_j - \epsilon \dfrac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$

For RMSProp, the sum of squared gradients is multiplied by a decay rate α and the current gradient – weighted by (1- α) – is added. The update step in the case of RMSProp looks the same as in AdaGrad. Here we divide the current gradient by the sum of the squared gradients to get the nice property of speeding up the updating of the weights along one dimension and slowing down the motion along the other.

Although SGD with momentum is able to find the global minimum faster, this algorithm takes a much longer path that could be dangerous. This is because a longer path means **more potential saddle points and local minima** of the loss function that could lie along that path. RMSProp, on the other hand, goes straight to the global minimum of the loss function without taking a detour.

1. **Handling Non-stationary Objectives**:
   - **RMSProp** is particularly well-suited for non-stationary objectives (where the data distribution changes over time), as it can adjust more dynamically to the changes compared to Adagrad.
2. **Empirical Performance**:
   - In practice, RMSProp often performs better than Adagrad on a variety of machine learning tasks. It tends to converge faster and reach better solutions, especially when dealing with deep learning models.

Overall, RMSProp is generally preferred for its ability to maintain a more stable and effective learning rate throughout training, leading to better performance on many complex tasks.

9c) Describe the saddle point problem in machine learning.

Key Characteristics of a Saddle Point:

1. Zero Gradient:

- At a saddle point, the gradient of the cost function is zero. This means that the partial derivatives with respect to each parameter are all equal to zero.

2.Neither Minimum nor Maximum:

- Unlike a local minimum or maximum, a saddle point is a point where the cost function neither reaches a minimum nor a maximum value.

3. **Flat in Some Dimensions, Steep in Others:**

- The surface of the cost function is flat in certain dimensions (where the partial derivatives are zero) and steep in others. It creates a saddle-like shape.

4. **Challenge for Optimization Algorithms:**

- Optimization algorithms, such as gradient descent, can get stuck or converge very slowly near saddle points because the gradient is zero, and the algorithm may struggle to determine the right direction to move.

10b) What is the best optimization algorithm in machine learning?

Stochastic Gradient Descent algorithm