1. (1', 3')

x_1	x_2	class
1	1	+1
-1	-1	-1
0	0.5	-1
0.1	0.5	-1
0.2	0.2	+1
0.9	0.5	+1

$W^T x = 0$ and $w = [1, 1]$

iter 1

(i) $y = 1\times 1 + 1\times 1 = 2$, class $= +1$ → correctly classified.
(ii) $y = 1\times(-1) + 1\times(-1) = -2$, class $= -1$ → correctly classified.
(iii) $y = 1\times 0 + 1\times 0.5 = 0.5$, class $= -1$ → wrongly classified.
hence $w = w - x = [1-0, 1-0.5]$ (as wrongly classified class is -1)
$= [1, 0.5]$
(iv) $y = 1\times 0.1 + 0.5\times 0.5 = 0.35$, class $= -1$ → wrongly classified.
hence $w = w - x = [1-0.1, 0.5-0.5]$ (as wrongly classified class is -1)
$= (0.9, 0.0)$
(v) $y = 0.9\times 0.2 + 0\times 0.2 = 0.9$, class $= +1$ → correctly classified.
(vi) $y = 0.9\times 0.9 + 0\times 0.5 = 0.81$, class $= +1$ → correctly classified.

iter - 2

(i) $y = 0.9\times 1 + 0\times 1 = 0.9$, class $= +1$ → correctly classified.
(ii) $y = 0.9\times(-1) + 0\times(-1) = -0.9$, class $= -1$ → correctly classified.
(iii) $y = 0.9\times 0 + 0\times 0.5 = 0$, class $= -1$ → wrongly classified.
hence $w = w - x = [0.9-0, 0-0.5]$ (as wrongly classified class is -1)
$= [0.9, -0.5]$
(iv) $y = 0.9\times 0.1 + (-0.5)\times 0.5 = -0.16$, class $= -1$ → correctly classified
(v) $y = 0.9\times 0.2 + (-0.5)\times 0.2 = 0.08$, class $= +1$ → correctly classified.
(vi) $y = 0.9\times 0.9 + (-0.5)\times 0.5 = 0.56$, class $= +1$ → correctly classified.

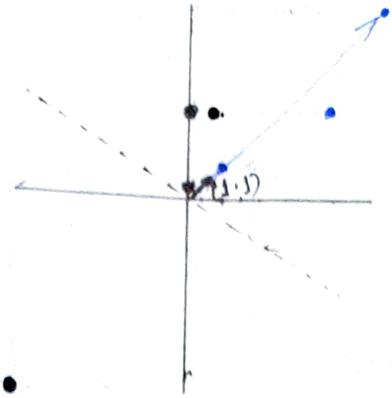
iter -3

(i) $y = 0.9\times 1 + (-0.5)\times 1 = 0.4$, class $= +1$, correctly classified.
(ii) $y = 0.9\times(-1) + (-0.5)\times(-1) = -0.4$, class $= -1$, correctly classified.
(iii) $y = 0.9\times 0 + (-0.5)\times 0.5 = -0.25$, class $= -1$, correctly classified.
(iv) $y = 0.9\times 0.1 + (-0.5)\times 0.5 = -0.16$, class $= -1$, correctly classified.
(v) $y = 0.9\times 0.2 + (-0.5)\times 0.2 = 0.08$, class $= +1$, correctly classified.
(vi) $y = 0.9\times 0.9 + (-0.5)\times 0.5 = 0.56$, class $= +1$, correctly classified.

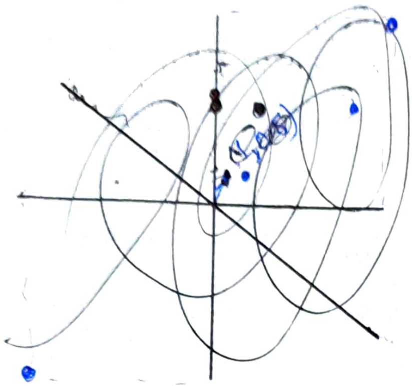
Therefore we can conclude in 3 steps the perceptron algorithm will converge.

1.2

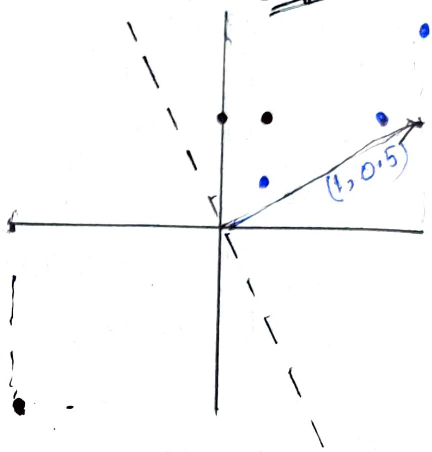
Initial.



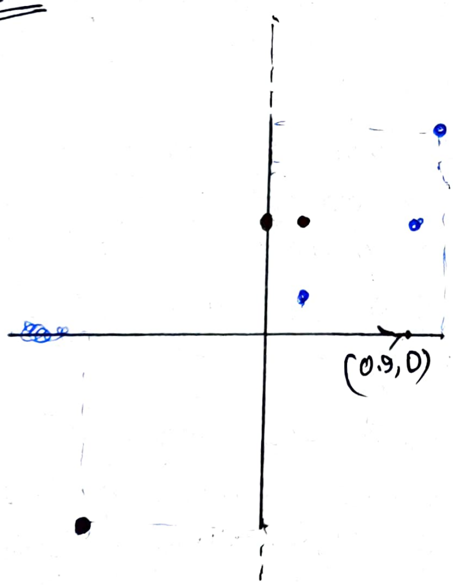
~~Step 1~~



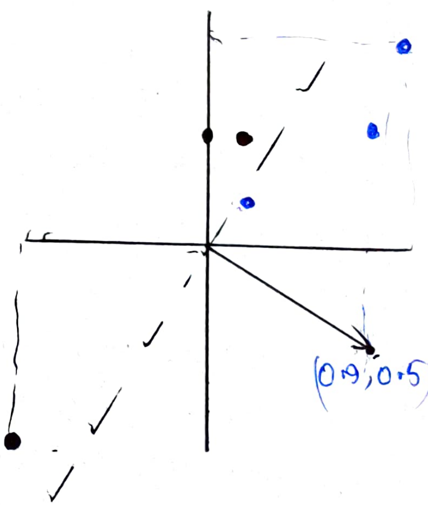
Step 1



Step-2



Step-3



The final ~~deicision~~ boundary ~~is~~ $w^T x = 0$

$\therefore \; [x_1, \; x_2] \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} = 0 \qquad \therefore \; 0.9x_1 + 0.5x_2 = 0$

$\therefore \; x_1 = 0.5, \; x_2 = -0.9.$

or $x_1 = -0.5, \; x_2 = 0.9.$

1.3

Lets consider the sets P and N are finite and linearly seperable.
Lets assume ~~the~~ we get the updated weighted vector w_t from perceptron learning algorithm in t no. of steps which is finite.
So if we prove t is finite then ~~the~~ we can conclude the perceptron learning algorithm will converge in finite no. of steps.

we can make three simplifacation without losing any generality.

(i) the sets P and N can be join in a single set $P' = P \cup N^-$
where N^- sets contains negated element of N.

(ii) The vectors in P' can be normalized, because if a weight vector w is ~~found~~ found so that $w \cdot x > 0$ is also valid for any vector γx where γ is a constant

(iii) now we can say w^* is normalized vector of w.

Lets consider after $t+1$ steps. weight vector w_{t+1} has been computed hence we can say at time t, a vector p_i was incorrectly classified by weight vector w_t, hence $w_{t+1} = w_t + p_i$

$\therefore$ angle between w_{t+1} and $w^* \Rightarrow \cos\phi = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|}$

we know $w^* w_{t+1} = w^*(w_t + p_i)$

$= w^* \cdot w_t + w^* p_i$

which is $\geq w^* t + \Delta$

with $\Delta = \min\{w^* p \mid \forall p \in P'\}$
as w^* creates an absolute linear seperation between P and N, $\Delta > 0$

$\therefore \quad w^* w_{t+1} \geq w^* w_0 + (t+1)\Delta$

again $\|w_{t+1}\|^2 = (w_t + p_i)(w_t + p_i)$

$= w_t^2 + 2 p_i w_t + p_i^2$

as $w_t * p_i$ $=0$ or negative.

$\|w_{t+1}\|^2 \leq \|w_t\|^2 + \|p_i\|^2$

$\leq \|w_t\|^2 + 1$

again $\|w_{t+1}\|^2 = \|w_0\|^2 + (t+1)$ as p_i is normalized

$\therefore \cos\phi = \dfrac{w^* w_0 + (t+1)\Delta}{\sqrt{\|w_0\|^2 + (t+1)}}$

$\dfrac{w^* w_0 + (t+1)\Delta}{\sqrt{\|w_0\|^2 + (t+1)}}$ grows proportionaly o $\sqrt{t}$

As Δ is positive, it can be arbitarily large, but ~~cosϕ is bound~~ + $\cos\phi \leq 1$. so t must be bounded by a maximum value which is finite.

3.

(i) **Sigmoid:**

$\sigma(x) = \dfrac{1}{1+e^{-x}}$

$$\frac{d}{dx}(\sigma(x)) = \frac{d}{dx}(1+e^{-x})^{-1}$$

$$= (1+e^{-x})^{-2}\left[\frac{d}{dx}(-e^{-x}) + \frac{d}{dx}(1)\right]$$

$$= (1+e^{-x})^{-2} \cdot -(e^{-x} \cdot -1) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= \sigma(x)(1-\sigma(x)).$$

(ii) **tanh:**

$\sigma(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

$$\frac{d}{dx}\left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right) = \frac{e^x + e^{-x}}{(e^x+e^{-x})^2}\frac{d}{dx}(e^x - e^{-x}) - \frac{e^x - e^{-x}}{(e^x+e^{-x})^2}\cdot\frac{d}{dx}(e^x + e^{-x})$$

$$= \frac{(e^x+e^{-x})(e^x+e^{-x}) - (e^x-e^{-x})(e^x-e^{-x})}{(e^x+e^{-x})^2}$$

$$= 1 - \left(\frac{e^x - e^{-x}}{e^x+e^{-x}}\right)^2 = 1 - (\tanh(x))^2$$

Relu:

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$\therefore f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

3.2 Strategies to avoid over-fitting in neural network:

Strategies are

(i) Simplifying the model or the decrease the complexity by reducing hidden layer and by creating smaller network

(ii) Early stopping

(iii) By using data augmentation

(iv) By using regularization

(v) by using dropouts.

3.3.

$f(z) = z_1 x + z_2 y$. $\quad x = [1,1]^T, \; y = [1,1]^T$

$f: R^2 \to R^2 \quad f(z) = z_1 x + z_2 y$

$z = g(r) = [r^2, r^3]$

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial z} \times \frac{\partial z}{\partial r} = \begin{bmatrix} x \\ y \end{bmatrix} [2r, 3r^2]$$

$$= \begin{bmatrix} 2r^2 x_1 & 3r^2 y_1 \\ 2r^2 x_2 & 3r^2 y_2 \end{bmatrix} = \begin{bmatrix} 8 & 12 \\ 8 & 12 \end{bmatrix}$$

$x = [1,1]$
$y = [1,1]$
$r = 2$

3.4.

(i) **Mean Squared Error:**

Mean squared error is defind by the following equation.

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

(ii) **Binary cross entropy:**

~~BSI~~

Binary cross entropy is defined by the below equation

$$BSE = -\frac{1}{n} \sum_{i=1}^{n} y_i \cdot \log(p(y_i)) + (1-y_i) \cdot \log(1-p(y_i)).$$

where $p(y)$ is the probablity of expected result

(iii) **Catagorical Cross Entropy:**

Categorical Cross entropy is defined by the following equation.

$$CC = -\sum_{i=1}^{n} p(x) \log(p(x))$$

where $p(x)$ is the probablity distribution of x.

3.5.

(i) **Batch Gradient Descent:**

Batch gradient descent takes all the training data into consideration to take a single step. We take the average of the gradients of all training ~~examples~~ examples and then use that mean gradient to update our parameters. So that's one step of gradient descent in one epoch.

3.5.

(ii) Stochastic Gradient Descent:

In Stochastic Gradient Descent we consider just one example at a time to take a single step. We do the following steps in one epoch for SGD.

(i) Take an example

(ii) Feed it to neural network.

(iii) Calculate its gradient

(iv) Use ~~gra~~ calculated gradient above to update the weights

(v) Repeat all steps above for all examples in the ~~traing~~ ~~step~~ ~~do~~ training dataset.

(iii) Mini-Batch Gradient Descent:

For mini Batch Gradient descent we take a batch of a fixed no. of training examples which is less than the actual dataset called a mini-batch. and do the following steps.

(i) Pick a mini batch

(ii) Feed it to the neural network.

(iii) Calculate the mean gradient of the mini batch.

(iv) Use the ~~mean~~ calculated mean gradient to update the weights.

(v) Repeat all above steps for all mini batches.

3.6.

(iv) Momentum based Gradient Descent:

Momentum based gradient Descent is ~~the~~ ~~the~~ calculate the exponentially weighted average of the gradients and use them instead of updating the weights. Which works faster than regular algorithm for gradient descent.

3.5

(v) Adam Solver:

Adam solver can be defined as the combination of RMSprop and ~~Stochastic~~ Stochastic Gradient descent with momentum. ~~It~~ uses squared gradients to scale the learning rate like RMSProp also takes advantage of momentum by moving average of the gradient descent instead of gradient.

2.2.

(ii) Change Loss function:

As we change the loss function from simplest ~~to the error~~ of Mean squared Error to the complex of Cross entropy loss we can observe ~~the~~ ~~that~~ that we get a better accuracy for less no. of epochs as it provide a better back propagation and ~~...~~ the network converges at a less no. of steps. where ~~as dee~~ as simpler loss function model underfits and we get less accuracy. But with increase in epochs we can observe that the simpler loss function work better than the complex loss function because that time it overfits the model.

2.2. (ii) Change in learning rate:

Keeping other parameter fixed if we ~~vary~~ change the learning rate we can observe ~~the~~ that for the less no. of epochs if learning rate is high it provide a decent accuracy because with higher learning rate the ~~the~~ network converges at a higher speed and fits the model but with higher no. of ~~accura~~ epochs if we increse the learning rate initially ~~it~~ the accuracy increases and after a ~~ce~~ value with increase in the learning rate accuracy decreases as the model misses its local minima ~~as~~ and overfits.

2.2: (iii) Change in Number of hidden layers:

Keeping other parameter fixed ~~if we increas~~ with less no. of hidden layers or minimal no. of hidden layers we can observe less accuracy ~~as~~ its a very simple neural network but as we ~~increa~~ increase the hidden layers we can observe ~~an~~ improvement in accuracy as the complexity of the network increases. But again after a certain no. of layers. if we increase the no. of layers accuracy decreases as its too complex and information flow doesn't reach till the end layers. and we didn't get optimum result.