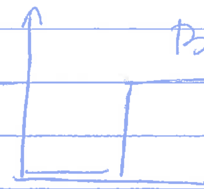
$2\frac{1}{2}$

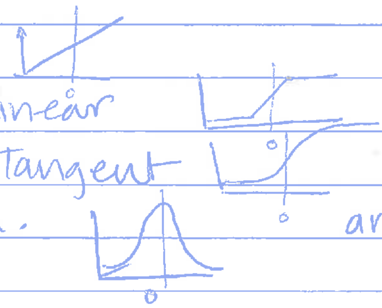
Components of a neural network

Activation Function

Unit Step Activation Function




* Linear
* Saturated Linear
* Hyperbolic Tangent
* Gaussian.

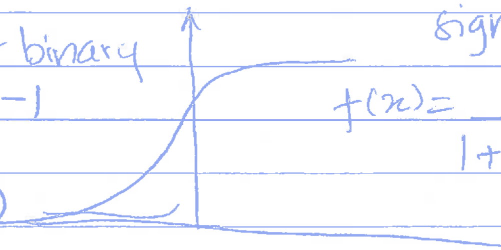


Not binary

any 0-1

↓

Radial Basis Function (RBF)



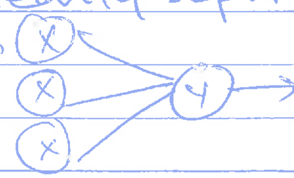
logistic sigmoid.

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

Network Topography

Single-layer network → linearly separable data

∘ Feed forward → Multilayer Perceptron (MLP)




∘ Deep Neural Network (DNN)

∘ Recurrent Network (both dir$^n$ flow of signal)

Delay → understand seq of data over time.

# Use fewest nodes that result in adequate performance

| STRENGTH | WEAKNESS |
|---|---|
| * Classification / numeric | ∘ Computationally intensive. |
| * Modelling complex patterns better than many other algos | ∘ Prone to overfitting. |
| * Makes few assumptions about the data. | ∘ Complex Black Box Model that are difficult to understand. |

Learning Rate: Rate of gradient descent

↑ rate ↓ training time.

Forward then Backward to check error – Epoch

# Neural Networks



| | X Hous study / Hour sleep | (y) test score |
|---|---|---|
| Train | (3, 5) | 75 |
| | (5, 1) | 82 |
| | (10, 2) | 93 |
| Test | (8, 3) | ? |

supervised
regression
if letter grade then classification

Usually good with Image recognition, music, text classification etc.
NPL

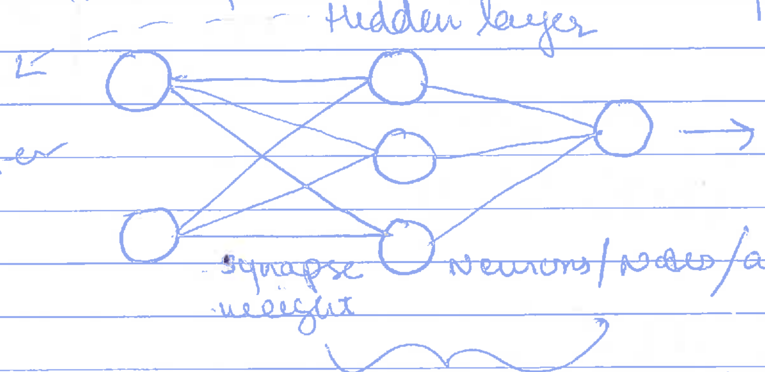
* Need to account for difference in scale / units of the data.

↳ Normalize / Scale the data

Two inputs . One output ( based on dimension of data)

Hidden layer

Hyper parameter



→ $\hat{y}$ (estimate)

synapse weight

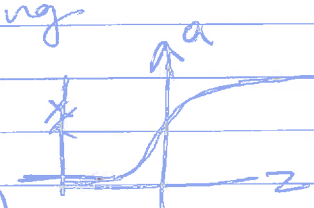
Neurons / Nodes / aggregate & apply activation function . (a)

Many hidden layer
↳ Deep Learning

$(z) = \sum x_i$

(a) activation function $= \dfrac{1}{1+e^{-z}}$

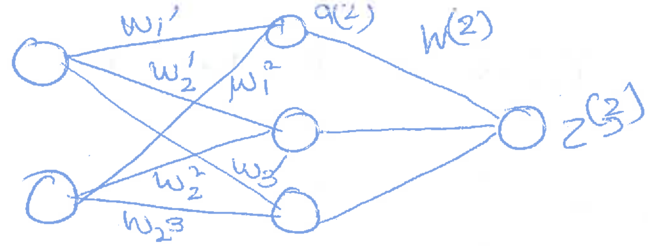
(sigmoid fn)





Structure & behavior of network is defined by Hyperparameter not updated as we train the network.
What is learned are the weights

$$\begin{bmatrix} 3 & 5 \\ 5 & 1 \\ 10 & 2 \end{bmatrix}$$




$$\begin{bmatrix} 3 & 5 \\ 5 & 1 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix} = \begin{bmatrix} 3W_{11}+5W_{21} & 3W_{12}+5W_{22} \\ 5W_{11}+1W_{21} & 5W_{12}+1W_{22} \\ 10W_{11}+2W_{21} & 10W_{12}+2W_{22} \end{bmatrix}$$

① --- $X$ $(3\times2)$    $W^{(1)}$ $(2\times3)$    $Z_1^{(2)}$ (activity of $2^{nd}$ layer) $(3\times3)$

Next we apply the activation function ~~g~~ on each value in $z^{(2)}$

$$a_{11} = \frac{1}{1+e^{-(z_{11})}}$$

sigmoid function is r $(3\times3)$

② --- $a^{(2)} = f(z^{(2)})$    $\underset{3\times1}{W^{(2)}} = \begin{bmatrix} W_{11} \\ W_{21} \\ W_{31} \end{bmatrix}$

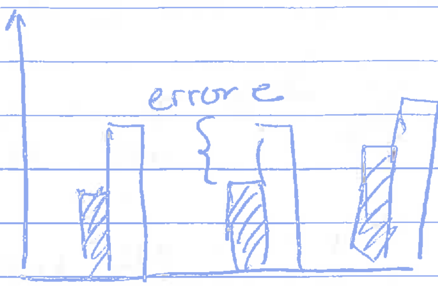
③ --- $\underset{(3\times1)}{z^{(3)}} = \underset{(3\times3)}{a^{(2)}} \; \underset{(3\times1)}{W^{(2)}}$

④ --- $\hat{y} = f(z^{(3)})$

<Next we will train>

# GRADIENT DESCENT

What we predict vs Actual




$\text{error} = y - \hat{y}$

cost function $\sum \frac{1}{2}(e_1^2 + e_2^2 + e_3^2)$

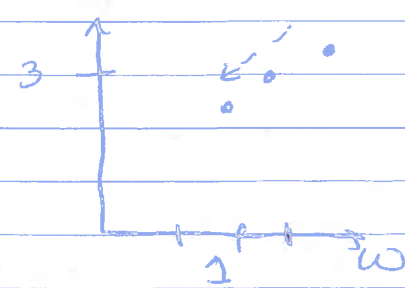
$J = \sum \frac{1}{2} e_i^2$

Training → minimizing the cost function (residual error)

cost function → ~~examples~~ inputs & weights
we cannot change the inputs
so we change the weight to
minimize cost

"CURSE OF DIMENTIONALITY"
to avoid calculating weights (all 9) we
look at w and calculate cost

For example in 1D, if
if know which way the
error is decreasing we can
go in that direction

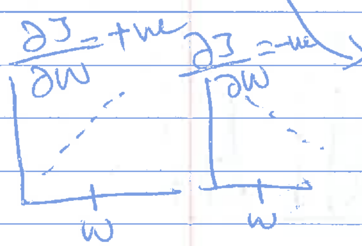



"Neumerical Estimation"

BUT...

Lets look at our equations

$$J = \sum \frac{1}{2}\left(y - f\left(f(XW^{(1)}) * W^{(2)}\right)\right)^2$$




Rate of change of J wrt $W^{(1)}$ & $W^{(2)}$ will tell us which way is down

We consider partial derivative → one weight at a time

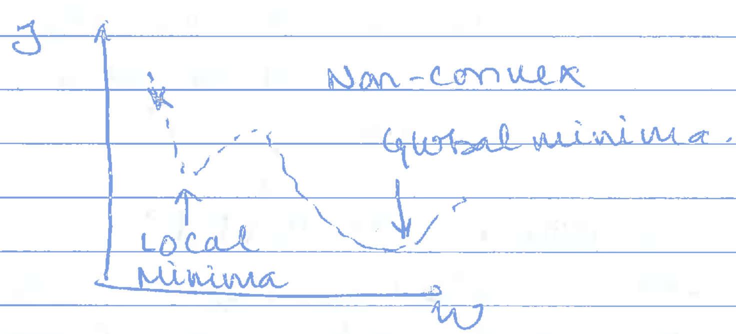
$$\frac{\partial J}{\partial W}$$

Take steps iteratively till cost stops decreasing

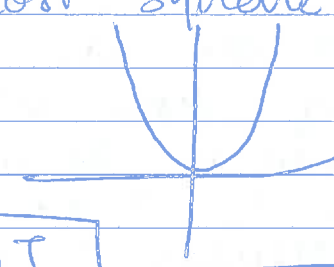
↳ GRADIENT DESCENT

Works well when high dimension data.

Problem:




By making the cost square we make this convex



Sto-castic

STOCHASTIC GRADIENT DESCENT

one example at a time to calculate $\frac{\partial J}{\partial W}$

input

and move in that direction

sometimes one at a time does not get us stuck in local minima

Batch Gradient Descent (adds the cost of one at a time)

$$\frac{\partial J}{\partial W_2} = \frac{\partial \sum \frac{1}{2}(y-\hat{y})^2}{\partial W^2}$$

$$= \cancel{\sum} \frac{\partial \frac{1}{2}(y-\hat{y})^2}{\partial W^2}$$

$$= \cancel{\sum}\ 2 \times \frac{1}{2}\ (y-\hat{y}) \frac{\partial \hat{y}}{\partial W^2}$$

$\hat{y} = f(z^{(3)})$

$$= \cancel{\sum} - (y-\hat{y}) \frac{\partial \hat{y}}{\partial z^3} * \frac{\partial z^3}{\partial W^2}$$

(drop)

$$f(z) = \frac{1}{1+e^{-z}}$$

$$f'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

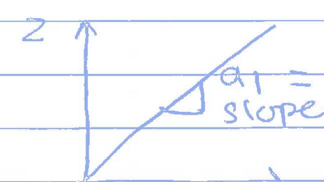
$z^{(3)} = a^{(2)} W^{(2)}$

(drop)

$$\frac{\partial J}{\partial W^2} = \sum - (y-\hat{y})\ f'(z^3)\ \frac{\partial z^3}{\partial W^2}$$

$$\begin{bmatrix} 3\times 1 \\ \\ \end{bmatrix} \begin{bmatrix} 3\times 1 \\ \\ \end{bmatrix} \begin{bmatrix} a^2 \\ 3\times 3 \end{bmatrix}$$

$\delta^{(3)}$




if we look $W_{11}$ at one synapse z & w have a linear relation

when we have to do the summation we can transpose $a^2$

$$a^T \delta^{(3)} = \frac{\partial J}{\partial W_2}$$

$$\delta^{(3)} = -(y-\hat{y})\ f'(z^3)$$

$$\frac{\partial J}{\partial w_1} = \sum \frac{1}{2}(y-\hat{y})^2$$

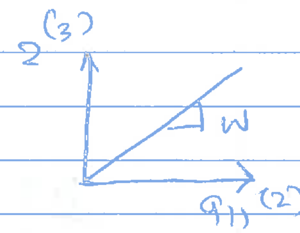
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w^{(1)}}$$

$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial z^3} \cdot \frac{\partial z^3}{\partial w^{(1)}}$$

$$= -(y-\hat{y})\, f'(z^{(3)}) \frac{\partial z^3}{\partial w^1}$$

$$= \delta^{(3)} \frac{\partial z^{(3)}}{\partial (w^1)}$$

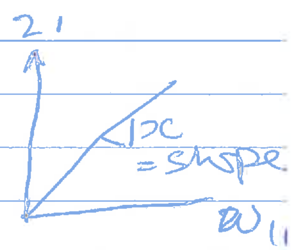
$$= \delta^{(3)} \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial (w^1)}$$


$$= \delta^{(3)} (W^{(2)})^T \frac{\partial a^{(2)}}{\partial w^{(1)}}$$

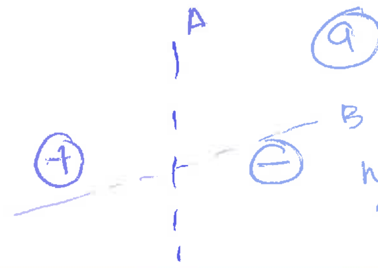
$$= \delta^{(3)} (W^{(2)})^T \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(1)}}$$

$$= \delta^{(3)} (W^{(2)})^T f'(z^2) \cdot \frac{\partial (z^{(2)})}{\partial w^{(1)}}$$




$x^{(T)}$

$$\frac{\partial J}{\partial w_1} = x^{(T)} \underbrace{\delta^{(3)} (W^{(2)})^T f'(z^2)}_{\delta^{(2)}}$$

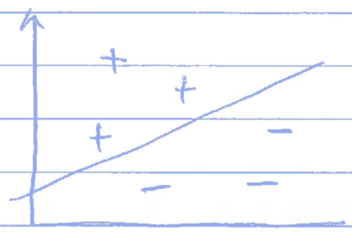


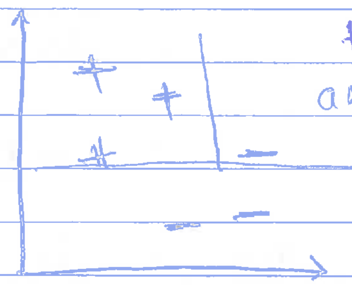

which is better?

# Support Vector Machines.
(Supervised)

Draw boundries.

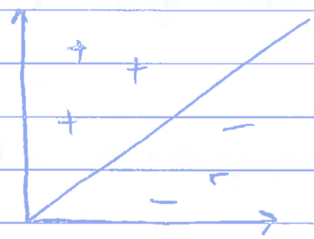
neural net





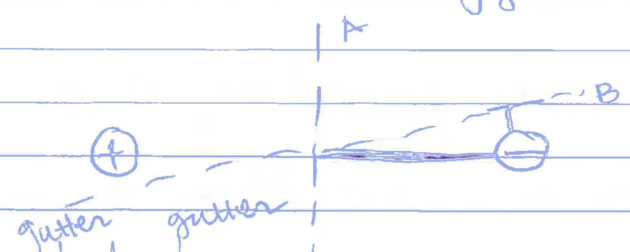
trees.
axis parallel splits

SVM



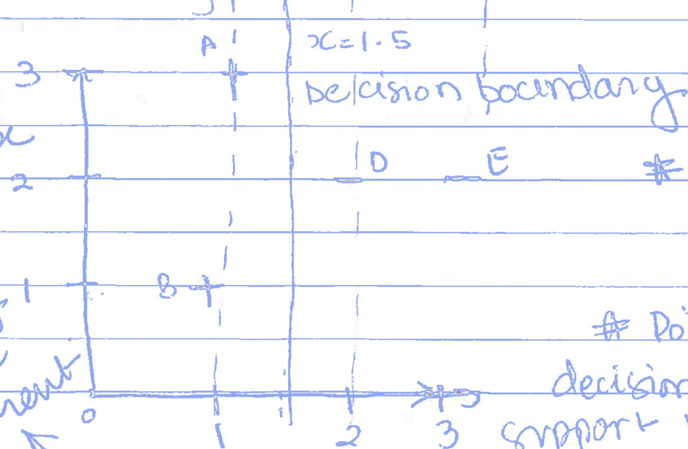
SVM separates the data with the largest possible margins. (gutter)

- SVM is a neumerical classifier → data has neumerical features.
- Draws a single decision boundry to maximize the margin width ~~or gutter~~. (b/w + & − points)
- It is a binary classifier. (+/−)
  w/a neural nets classify as (1/0)




Distance to traing point is max hence (A)

A, B, D are support vectors because if we remove any the boundry would be different




* Cannot be in the gutter but can be on the gutter.
* Points necessary to define the decision boundary are called SV
support Vectors

4 training points

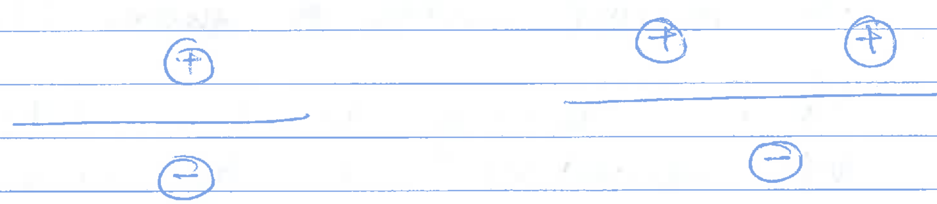
2 classes of points

3 support vectors

Margin is the area between the two gutters.
Margin width is the distance from one gutter to another
Margin width = 1

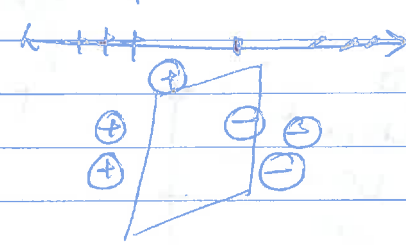
We can add points anywhere else → no change

add points on the gutter → Are those support vectors
so if we add the ~~points~~ such that if we remove them then the boundary would change → the support vectors.



in 2D our boundary is a line (2/3 SV)

in 1D our boundary is a point (2 SV)



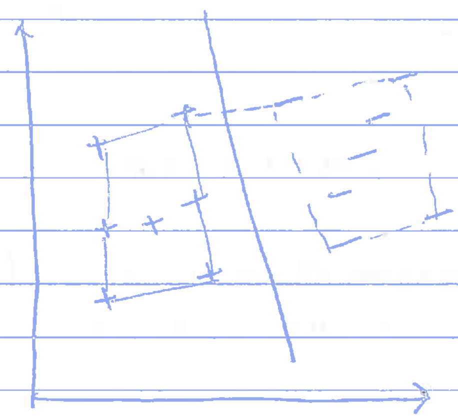
in 3D → plane

Always need at least 2 SVs.

Where does the decision boundary go?

We use the method called CONVEX HULL
then we search for Maximum Margin Hyperplane (MMH)
MMH selected is the one that creates the greatest separation.

* Identifying SVs allows to store the classification model very compactly even in case of large # of features.



Outer boundary is known as Convex Hull
- Find the shortest line between the two convex hulls.
- Bisector of that line is the Decision boundary.

$$\vec{w} \cdot \vec{x} + b = 0$$

weights $\vec{w} = \{ \vec{w}_1, \vec{w}_2, \dots \vec{w}_n \}$

$b$ = bias { intercept term }

goal is to find weights, such that

$$\left.\begin{array}{l} \vec{w} \cdot \vec{x} + b \geq +1 \\ \vec{w} \cdot \vec{x} + b \leq -1 \end{array}\right\}$$ weights that specify two hyperplanes.

Distance b/w two points $= \dfrac{2}{\|\vec{w}\|}$ → euclidean norm (distance from origin to vector)

To max distance, we need to min $\|\vec{w}\|$

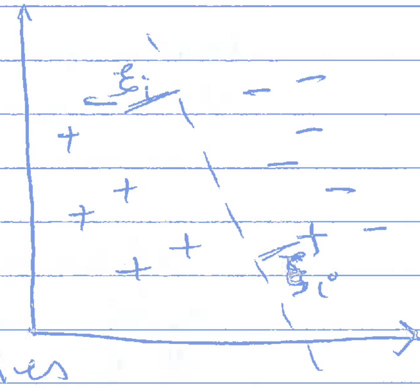
$$\min \frac{1}{2} \|\vec{w}\|^2$$

$$\text{s.t. } y(\vec{w} \cdot \vec{x}_i - b) \geq 1, \quad \forall \vec{x}_i \implies$$ subject to each of y is correctly classified

s.t. = subject to; $y$: +1 or −1; $\forall$ = for all

What if the data is not linearly separable?

⇒ Use of slack variable
points are allowed to be on the wrong side by defining a soft margin using slack

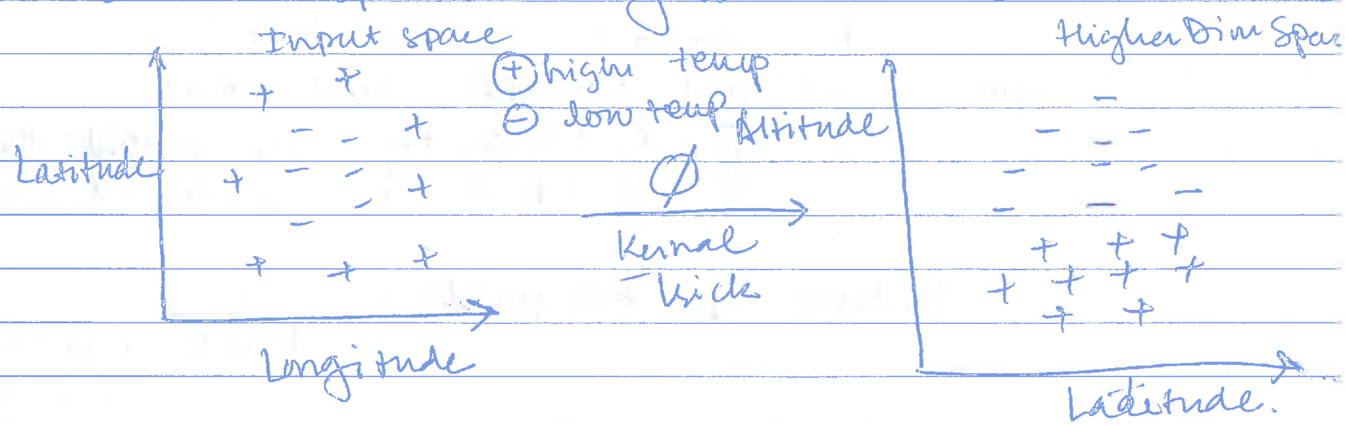



In this case a cost is applied to data points that voilate the constraint & the algorithm tries to minimize the COST instead of maximizing the MARGIN

Adjustment of the "COST PARAMETER" is hence unimportant

↑ CP ⇒ algo ~~strives~~ strives for 100% separation
↓ CP ⇒ algo strives for wider overall margin

⇒ Use of Kernel ~~trick~~ trick ⟨non-linear kernals⟩
Ability to map the problem into a higher dimension space using kernal trick




trick is to construct new features that express mathematically relationship b/w measured characteristics
↳ SVM learns concepts not explicity measured in the original data.

## Strength

1. Classification or numeric prediction.
2. Not prone to noisy data.
3. Not prone to overfitting.
4. May be easier to use than neural networks
5. High accuracy & wins in Data Mining Competitions

## Weakness

1. Finding best model requires testing of various combos of Kernals & Model parameters
2. Can be slow to train
3. Results in a complex black box that is difficult to understand.

## KERNAL FUNCTIONS

$$K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j})$$

$\phi$ - mapping data in another space.

Dot product

return single scalar number.

* LINEAR KERNAL

$$K(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$$

* POLYNOMIAL KERNAL ( simple non linear transformation)

$$K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^d$$

* SIGMOID KERNAL (similar to neural network activation function)

$$K(\vec{x_i}, \vec{x_j}) = \tanh(\kappa \vec{x_i} \cdot \vec{x_j} - \delta)$$

kappa delta.

* GAUSSIAN RBF Kernal

$$K(\vec{x_i}, \vec{x_j}) = e^{\frac{-\|\vec{x_i} - \vec{x_j}\|^2}{2\sigma^2}}$$

Example next .....