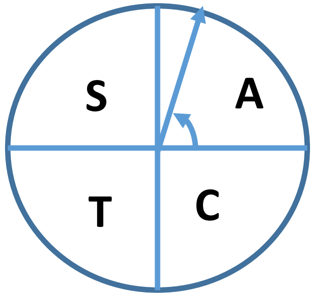
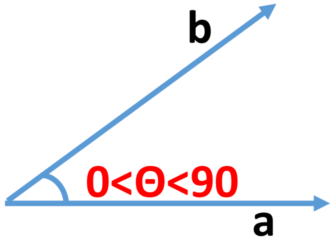
Under the hood – Why the perceptron rule works

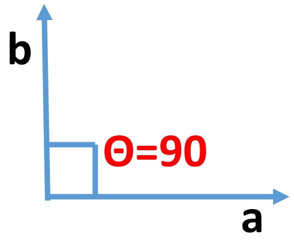
- Preliminaries:
  - Suppose two vectors ***a*** and ***b*** are separated by an angle **θ.** The inner product **a.b** is defined as **a.b =||a||||b||cos θ.**
    - L2 norm is always non-negative; -- it can either be positive or zero. This implies the sign of a.b can be inferred from the sign of cos θ. If cos θ is negative, a.b will be negative; if cos θ is positive, a.b will be positive.

All **S**tudents **T**ake **C**hemistry !!

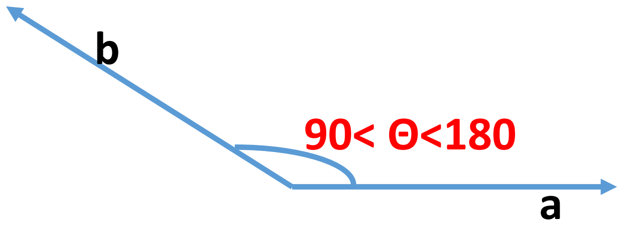




CosΘ>0 => a.b>0



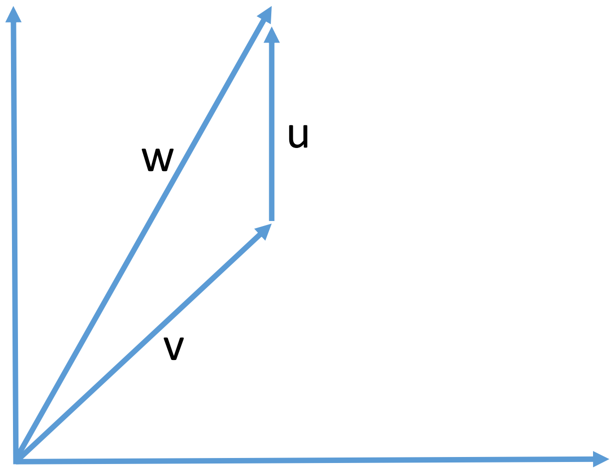
CosΘ=0 => a.b=0



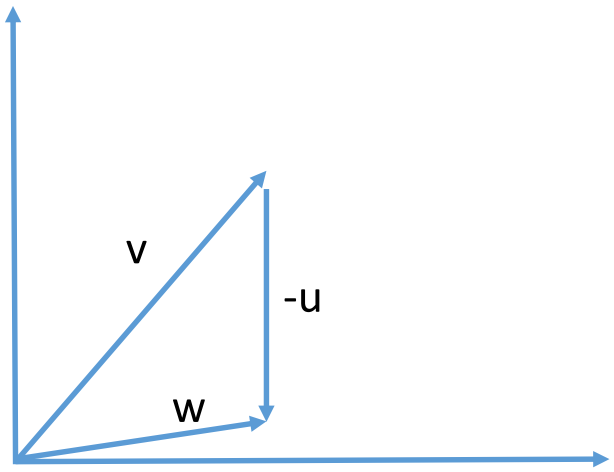
CosΘ<0 => a.b<0

Under the hood – Why the perceptron rule works

- Some more preliminaries …
- Consider two 2D vectors for simplicity. Geometrically, adding two vectors is equivalent to appending one vector to the end of the other.



w=v + u

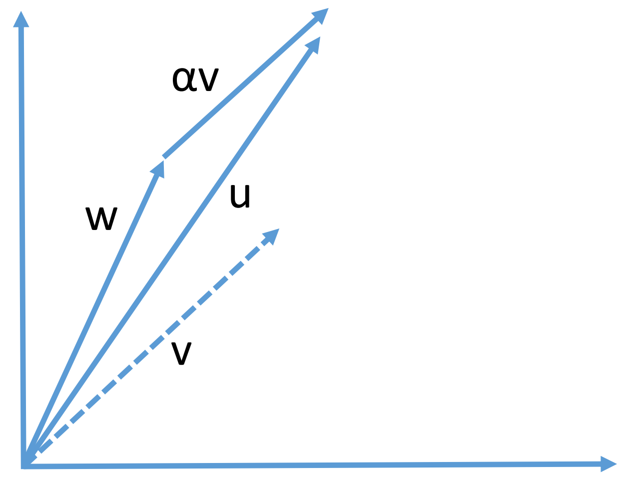


w=v + -u

Under the hood – Why the perceptron rule works

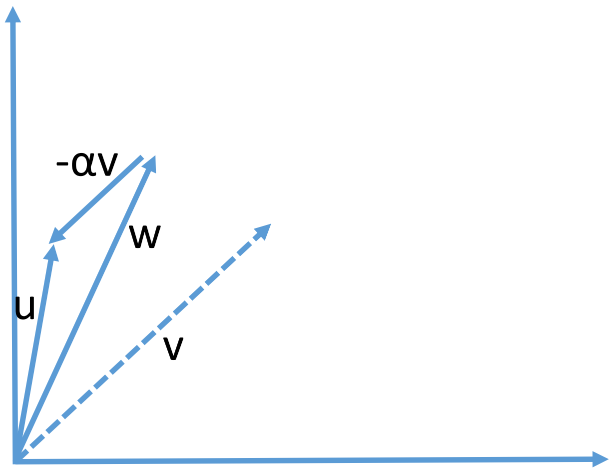
- Some more preliminaries …
- What if we want some guarantees on the result of the addition, e.g., if we want to add some vector to *w* and have the resultant vector (i) pointing more towards the direction of some vector *v*, (ii) pointing further away from the direction of some vector *v*

(i)



u=w+αv

(ii)



u=w+(-αv)

Under the hood – Why the perceptron rule works

- Recall the input to the activation function was:

$$\sum_{i=0}^{n} \boldsymbol{w_i x_i}$$

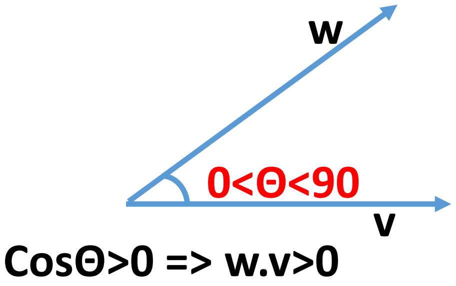
which for the 3D case, for example, would be equal to:

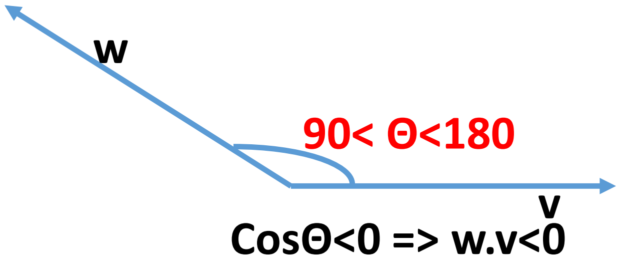
$\boldsymbol{w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3}$

- But this is the dot product between two vectors w=[$\boldsymbol{w_0}$ $\boldsymbol{w_1 w_2}$ $\boldsymbol{w_3}$] and x= [$\boldsymbol{x_0}$ $\boldsymbol{x_1}$ $\boldsymbol{x_2}$ $\boldsymbol{x_3}$]
- So, we basically have $\sum_{i=0}^{n} \boldsymbol{w_i x_i}$=$w.v$
- **So, during the perceptron learning process, all that is happening is a computation of a dot product between the weight vector and each input vector**

Under the hood – Why the perceptron rule works

- Lets analyze the cases where the algorithm gave us a wrong result
  - Case 1: Learning algorithm assigned a sample to the class label of 1 (i.e., *w.v* was >0) yet it should have belonged to class label 0 (i.e., *w.v* should have been <0).
  - In this case, it means that we had the weight and input vectors aligned as the figure to the left, yet correct classification would have required them to be aligned as the figure to the right



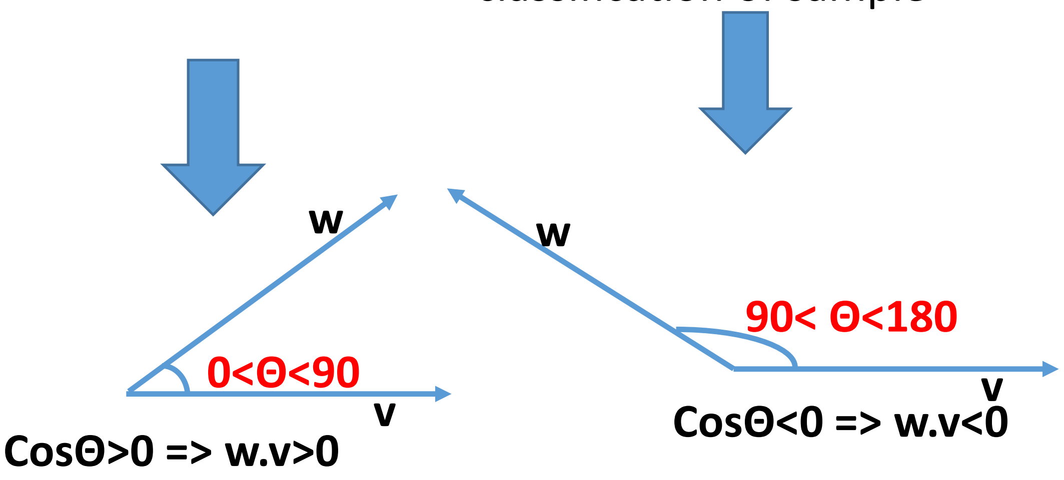


Under the hood – Why perceptron rule works

- Case 1 cont'd

What we have:

What we need for correct classification of sample



How do we fix Case 1?

✓ Rotate vector w away from vector v.

✓ That is, find a new value, w_{new} of w, such that: $w_{new}=w+(-\alpha v)$

Check: How did the perceptron algorithm fix Case 1?

✓ In our Case 1, $y_{classifier}=1$ (because the classifier finds that w.v>0). Since we know that the classifier fails to get the correct decision, this means $y_{target}=0$.
Recall that the error is $\delta=y_{target}-y_{classifier}$
So, that particular iteration would have $\delta = -1$.

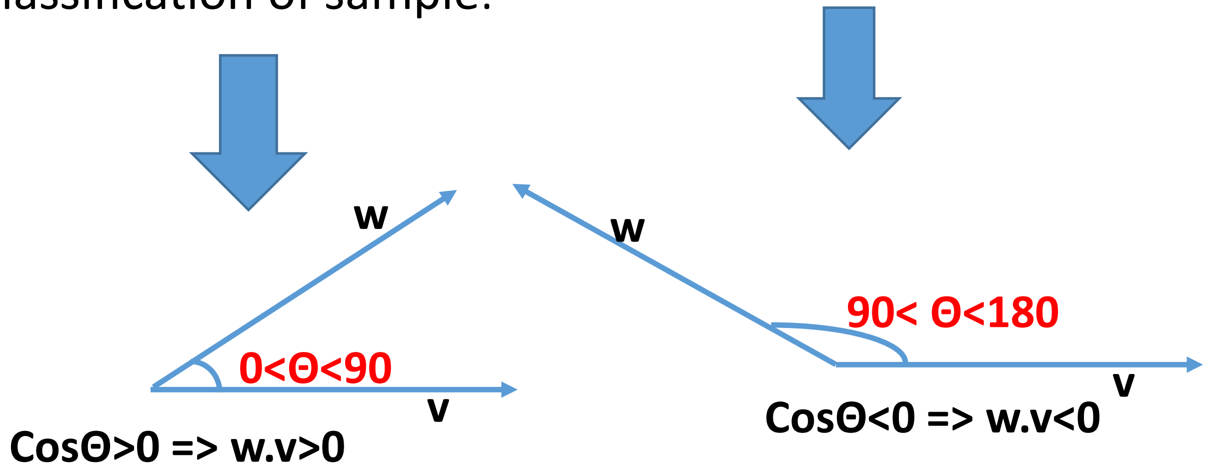
✓ And the weight update would be $\Delta w = \eta \times \delta \times x_{train}$ which is the same as $\Delta w = (-1)*\eta \times x_{train}$
The new weight is hence $w_{new}=w+(-\eta \times x_{train})$ which is in agreement with our strategy above for fixing Case1, since η is a positive constant between 0 and 1 and our v is x_{train}

Under the hood – Why perceptron rule works

- Lets analyze the cases where the algorithm gave us a wrong result
  - Case 2: Learning algorithm assigned a sample to the class label of 0 (i.e., *w.v* was <0) yet it should have belonged to class label 1 (i.e., *w.v* should have been >0).

What we need for correct classification of sample:

What we have:



This time we want to align w more towards the direction of v so as to work towards attaining the condition **w.v>0**

Using an idea similar to what we used in the previous case, we need to get $w_{new}=w+(\alpha v)$.
Easy to check that the perceptron algorithm did just this !

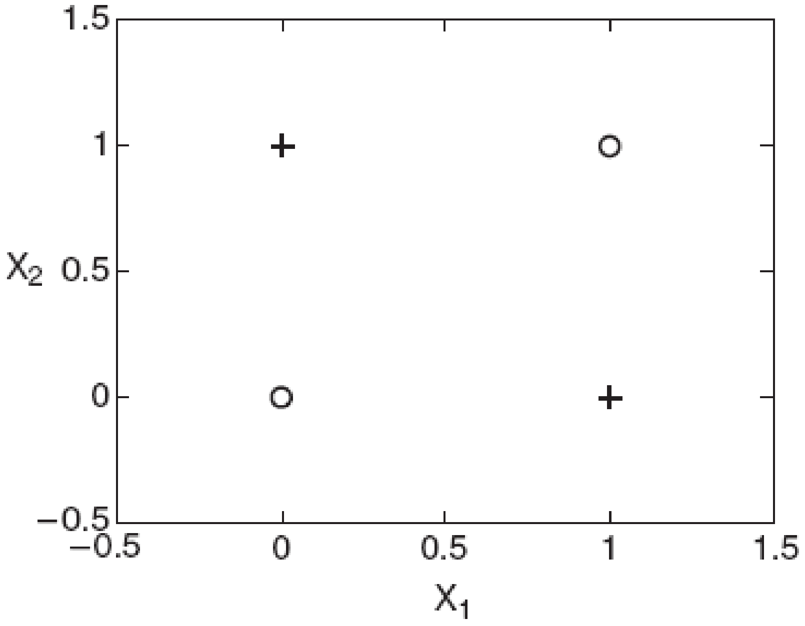
Simple Perceptron: some issues

- On test data, the perceptron will use the final weight vector obtained during training.
- But this may not necessarily be a good idea ! One of the "wrong" weight vectors produced before the final "correct" vector could actually perform better on test data
  - Averaged perceptron (average some of the well performing weights and predict)
  - Voted perceptron (vote on predictions of intermediate vectors)

Simple Perceptron: some issues

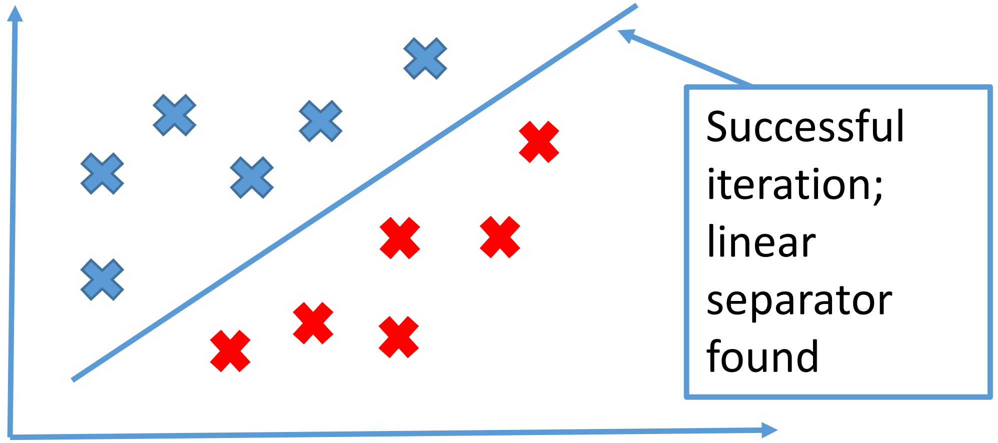
- Perceptron rule wont converge if classes not linearly separable.
- E.g., XOR problem (OR without AND)

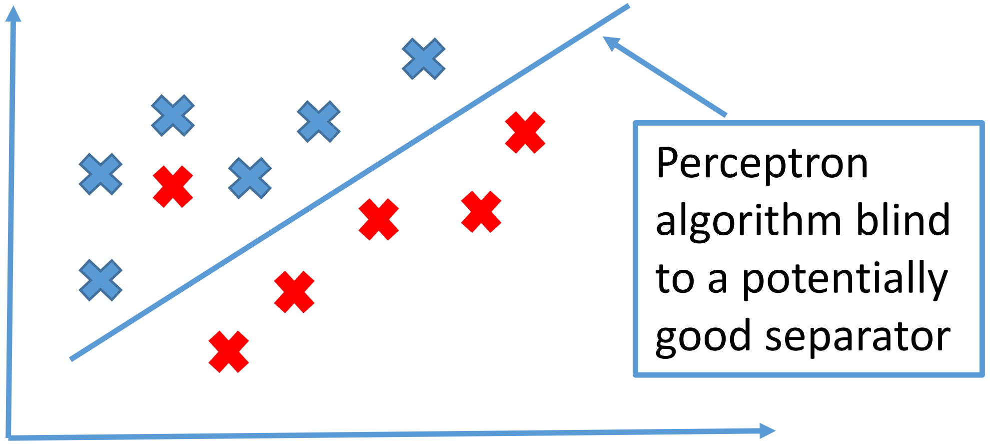
x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0



Multi-layer Perceptron and Back Propagation

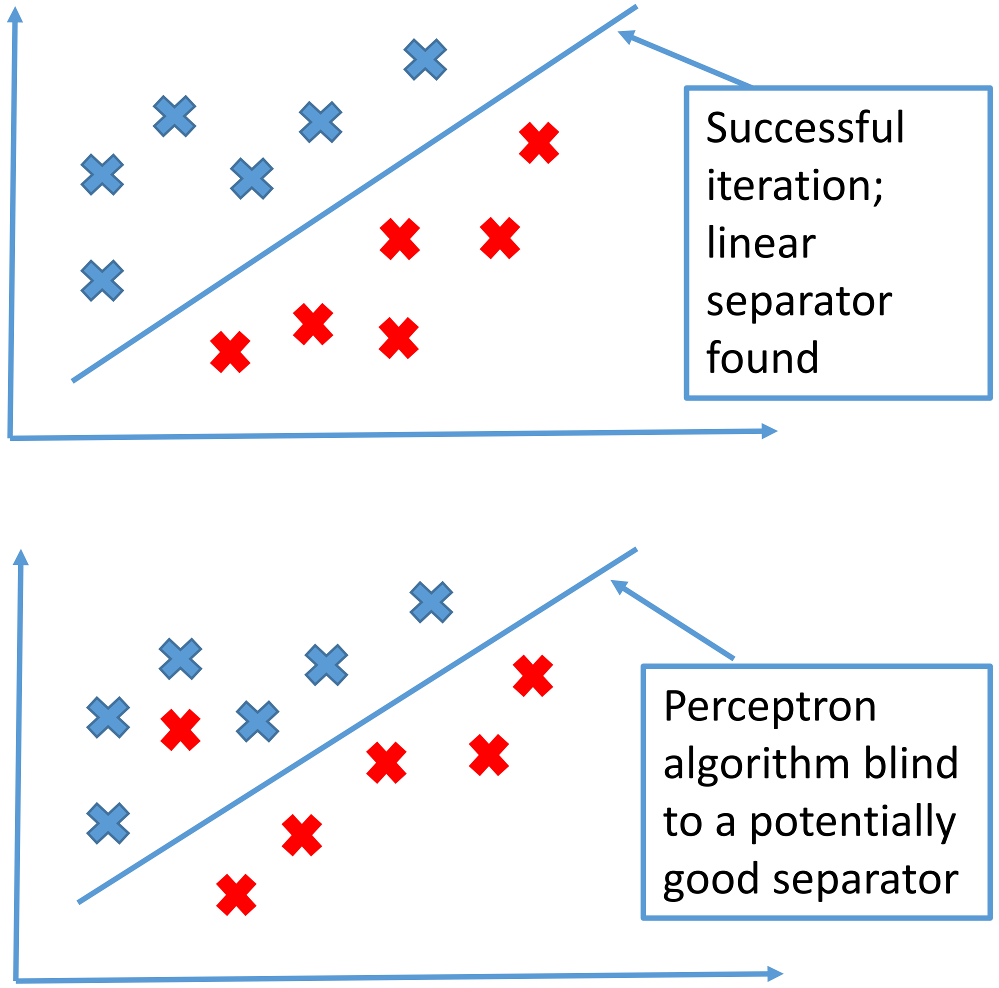
- The perceptron algorithm solves linearly separable cases
- Mechanism of algorithm is such that iteration ends only when linear separator is found
- What if the linear boundary doesn't exist, i.e., classes not linearly separable?





What if the linear boundary doesn’t exist ?

- A possible fix to this is an algorithm which optimizes some metric,
    - Algorithm will stop if metric is optimized regardless of whether a linear separator can be found or not.
  - Any ideas on candidate metrics?



What Error Metric?

- Define a measure of the difference between the network output and target vector
- This difference is then treated as an error to be minimized by adjusting the weights
- The weights corresponding to the minimum error define the classification boundary
- **What are the candidate error metrics?**

What Error Metric?

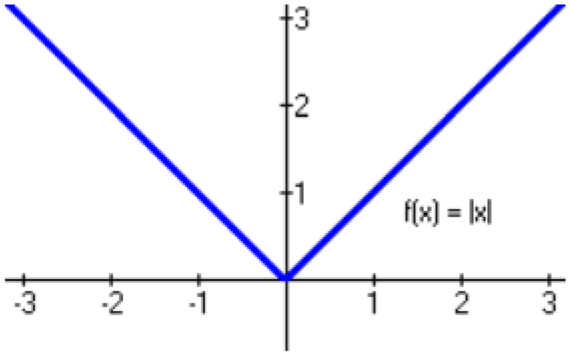
- If e^p is the error seen with training pattern p, the total error could be expressed as:

$$E = \sum_{p=1}^{N} e_p$$

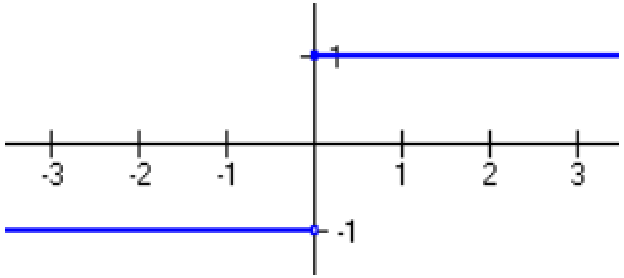
- **Possible options for defining e_p**:
- If y_p is the network's output for a pattern p, and t_p is the target output for this pattern, many possible error metrics can be defined between y_p and t_p. Examples:
  - $e_p = t_p - y_p$
  - $e_p = |t_p - y_p|$
  - $e_p = (t_p - y_p)^2$
  - Many more …

Why not minimize $\Sigma|e_i|$?

- This ensures that an error of -1 is treated similarly as that of 1
- But has its issues too …



Derivative Vs x



What Error Metric?

- The sum of squared errors is often (though not always) the metric of choice:

$E = \sum_{p=1}^{N} e_p$ where $e_p = (t_p - y_p)^2$

  - SSE is differentiable (smooth) everywhere, so allows required parameters to be easily estimated using gradient-based methods
- SSE is not without issues though
  - Suffers from outlier effects (for this reason it is said to lack robustness)
  - Data preprocessing for outlier removal is crucial when using it
- Our new strategy is to get the simple perceptron (or more complex network) to minimize the SSE error. **What are the candidate approaches for minimization of the error?**

How do we minimize the error metric?

- **Option #1: Straight Differentiation**

$$\boldsymbol{Example}\text{:}\ \boldsymbol{y = x^2 + 2x + 4}$$

$\frac{dy}{dx} = 2x + 2$; At turning point $2x + 2 = 0 \Rightarrow x = -1$

$$\frac{d^2y}{dx^2} = 2 \Rightarrow \frac{d^2y}{dx^2}(x = -1) = 2$$

Hence the point $(-1,3)$ *is a minimum*.

In general if $\frac{d^2y}{dx^2}(x = -1)$ had been negative, then $(-1,3)$ would be a maxima

Or if $\frac{d^2y}{dx^2}(x = -1)$ had been zero, then $(-1,3)$ would be a point of inflexion/saddle point

How do we minimize the error metric?

- **Option #1: Straight Differentiation**

Example: $f(x,y) = 3x^2 - 12x + 2y^2 + 16y - 10$

$$\frac{\partial f}{\partial x} = 6x - 12 = 0 \Rightarrow x = 2$$

$$\frac{\partial f}{\partial y} = 4y + 16 = 0 \Rightarrow y = -4$$

The point (2,-4) is a critical point

$$\frac{\partial^2 f}{\partial x^2} = 6; \frac{\partial^2 f}{\partial y^2} = 4; \frac{\partial^2 f}{\partial x \partial y} = 0 \ (all\ independent\ of\ x, y)$$

- D=6*4-0=24

Conclusion: (2,-4) is a relative minima

Would this option work for us ?

Second partial derivatives test:

(x_0,y_0) is a critical point of z=f(x,y) if:

$$\frac{\partial f}{\partial x}(x_0, y_0) = 0\ and\ \frac{\partial f}{\partial y}(x_0, y_0) = 0.$$

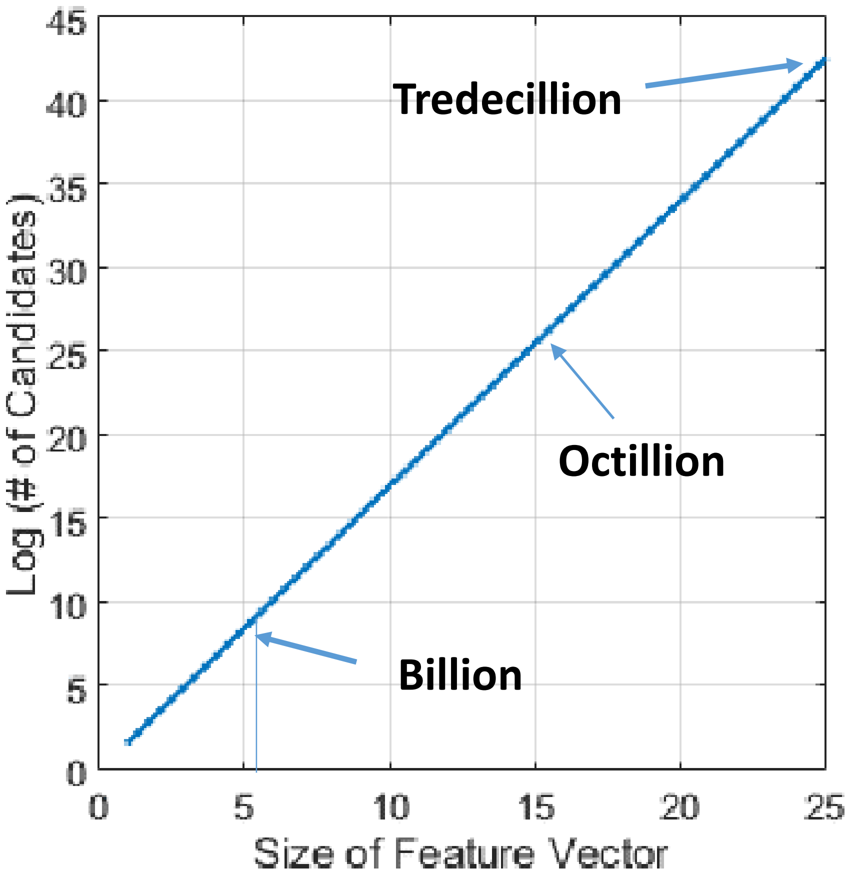
What kind of critical point is (x_0,y_0)?

Let D=$[\frac{\partial^2 f}{\partial x^2}(x_0, y_0) * \frac{\partial^2 f}{\partial y^2}(x_0, y_0)] - [(\frac{\partial^2 f}{\partial x \partial y}(x_0, y_0)]^2$

- (x_0,y_0) is a relative minimum value if D>0 and $\frac{\partial^2 f}{\partial x^2}(x_0, y_0) > 0$
- (x_0,y_0) is a relative maximum value if D>0 and $\frac{\partial^2 f}{\partial x^2}(x_0, y_0) < 0$
- If D<0, then (x_0,y_0) is a saddle point
- If D=0, we are unable to conclude anything; turning point could be max, min or saddle

How do we minimize the error metric?

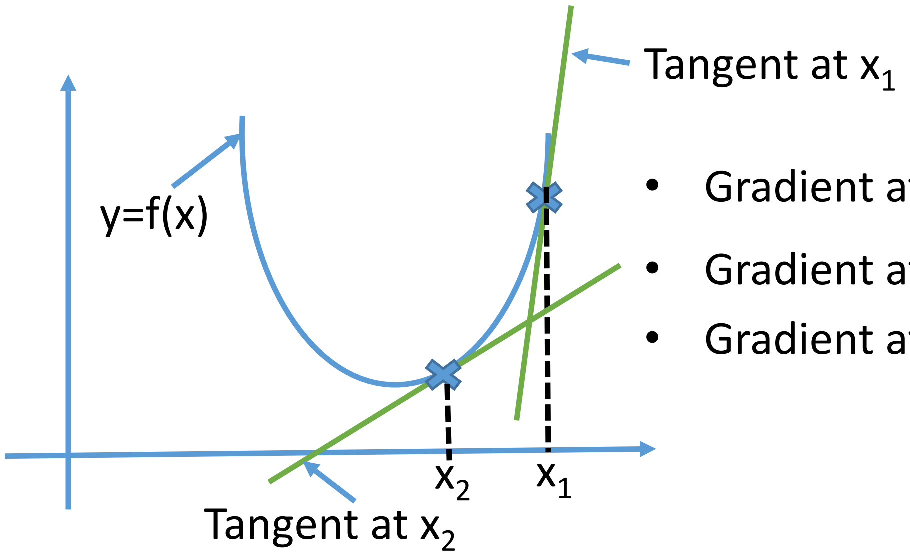
- **Option # 2: Brute force**
- Assume we have two independent variables x and y and the response variable z (the error metric)
- Example: Define a domain of x having 50 values and y having 50 values.
- We need 50^2 =2500 different combinations of x and y to get some estimate of where our minimum value of z lies.
- If we have 3 inputs, we will have to sift through 50^3 different candidate values …



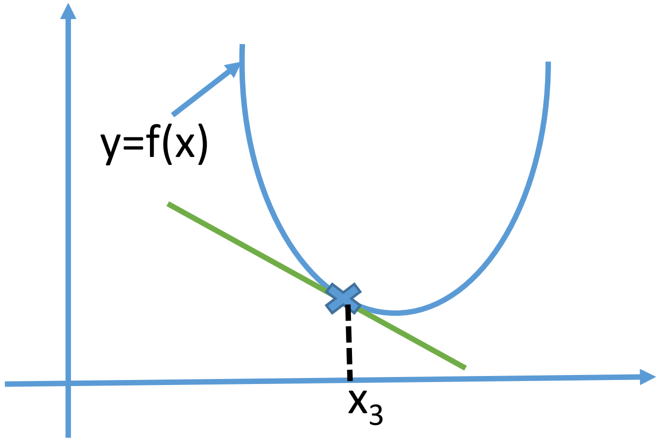
Would this option work for us ?

How do we minimize the error metric?- Iterative Approach: Gradient decent

- Gradient: Slope of the tangent of a function at a point.



- Gradient at $x_1 = \frac{dy}{dx}(x = x_1) > 0$
- Gradient at $x_2 = \frac{dy}{dx}(x = x_2) > 0$
- Gradient at x_1 exceeds that at x_2

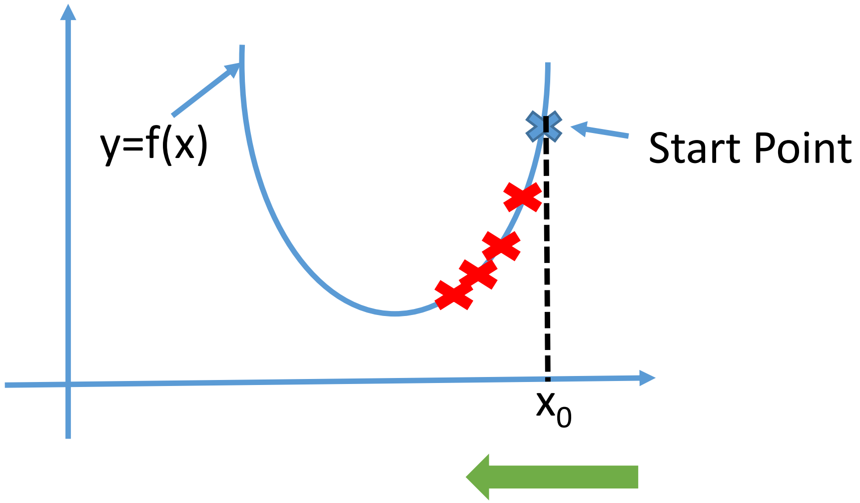


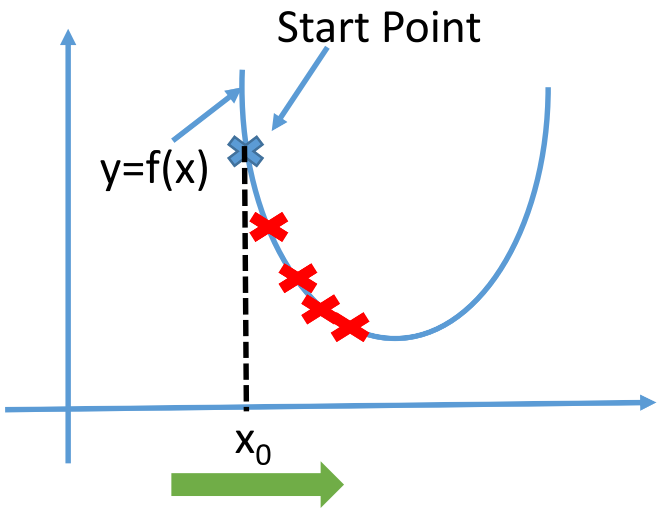
- Gradient at $x_3 = \frac{dy}{dx}(x = x_3) < 0)$

- Taking the single variable case, gradient decent method works as follows: $x_{j+1} = x_j - \alpha \frac{dy}{dx}(x = x_j)$

Gradient Decent

- How gradient decent searches for the minima (assuming a very small α).
- $x_{j+1} = x_j - \alpha \frac{dy}{dx}(x = x_j)$





Gradient decent – Some issues to take note of

- Very small α; the algorithm may take very long to converge
- Very large α
  - Algorithm may fail to converge
  - Or may diverge