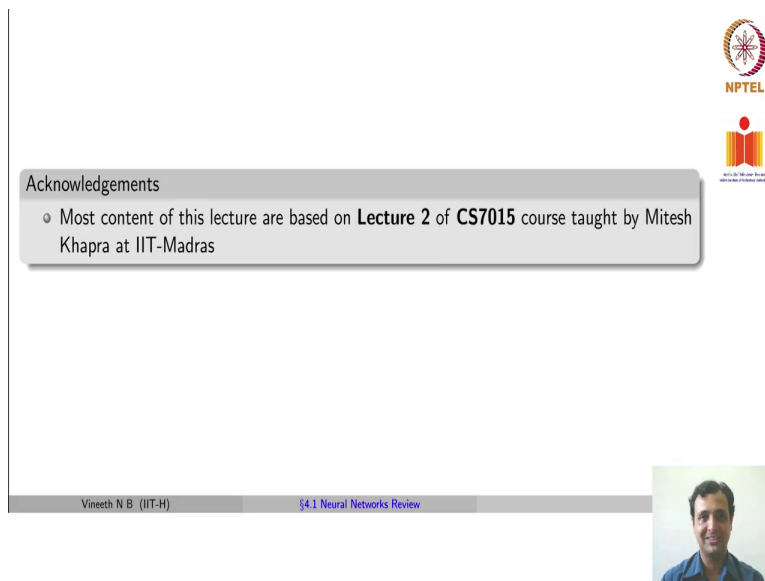


**Deep Learning for Computer Vision**  
**Professor Vineeth N Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Neural Networks: A Review – Part 01**

In the lecture so far we focused on computer vision as it was studied and practiced before the advent of deep learning. Deep learning has completely revolutionized how computer vision is done today. While there have been several dimensions to the efforts in computer vision before this advent of deep learning, hopefully the few lectures that we have had so far give you a peek into all those efforts, some of which continue to be relevant today when combined with deep learning especially.

Moving forward, we focus our lectures on deep learning and its use in computer vision. We will start with the first lecture as a review on neural networks, for those of you that may have done a deep learning or a neural network course before, this maybe a review, but even if you have not done such a course before all the interactive material will be covered in these lectures.

(Refer Slide Time: 01:25)



The slide features a central grey box with the following text:

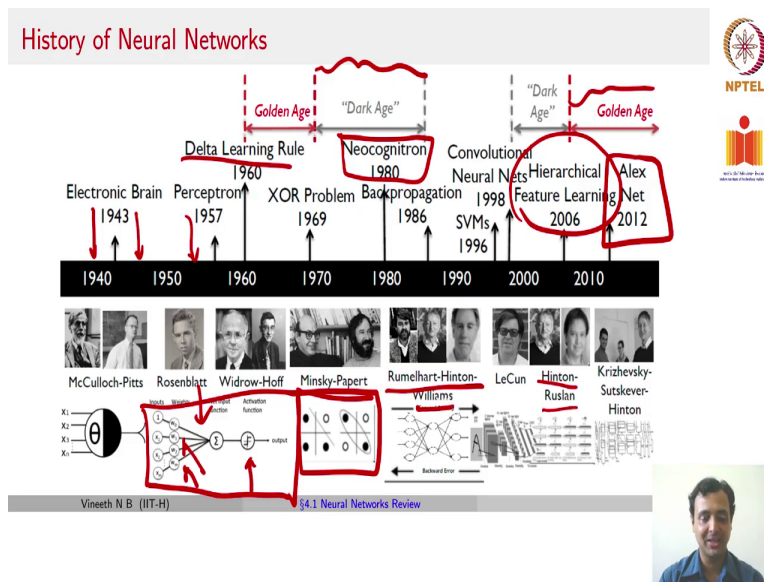
Acknowledgements

- Most content of this lecture are based on **Lecture 2** of **CS7015** course taught by Mitesh Khapra at IIT-Madras

Logos for NPTEL and IIT Madras are visible in the top right corner. A small portrait of a man is in the bottom right corner. The footer contains the text: Vineeth N B. (IIT-H) | 4.1 Neural Networks Review

I would like to acknowledge that most of the content of this lecture is based on the excellent lectures of Mitesh Khapra on deep learning at IIT Madras.

(Refer Slide Time: 001:35)



To start with the history of neural networks, the history of neural networks is perhaps older than the history of computing itself, it is started in the early 1940's when two researchers, McCulloch and Pitts work on developing the computational model of the human brain and being able to use that model to simulate simple function such as logic gates.

Later in the 1940's a psychologist named Donald Hebb, who came up with what is known as Hebbian learning which is used to this day. Hebbian learning states that in the human brain two neurons that are wired together, fired together two neurons that are out of sync fail to link, rather if two neurons keep getting fired together the strength of the connection between these two neurons gets better and better over time.

While if there are two neurons that rarely get fire together the strength of their connection becomes weaker and weaker over time, this is a principle that is used to this day. Later in the 1950's Frank's Rosenblatt came up with the first model of the neural network known as the perceptron. The perceptron is the simplest neural network one could have, which has a set of neurons which is known as a layer of neurons, which receives a set of inputs, acts on them through a set of weights and then gives an output which is thresholded to get the final decision. Rosenblatt also proposed a learning algorithm to obtain the weights in this simple neural network.

In the early 1960's this learning rule was improved by Widrow and Hoff which is known as the Widrow-Hoff learning rule or the Delta learning rule. This was the foundation of algorithms that we use to this day, which we will talk about. However, Rosenblatt claimed that this perceptron could classify any kind of a data setting, a binary classification setting, Minsky and Papert two researchers at that time disprove this using the example of the XOR gate which clearly cannot be separated by a simple passive drop.

This led to a downfall of neural networks in the 1970's because all the hype of the perceptron has simply been crashed because of its inability to handle the XOR scenario as well as similar complex functions. In the 1980's the interest once again revived with development of Neocognitron in 1980, which is largely considered as the first version of a convolutional neural network and in 1986 was the development of a ground breaking algorithm back propagation which we use to this day by Rumelhart Hinton and Williams.

Later in the 1980's and the mid 1990's versions of neural networks such as convolutional neural networks were developed at that time and these neural networks were adopted in various different applications. One popular application at that time was the handwritten digit recognition problem which is today known as the MNIST data set, the reason for the popularity of this data set was the application setting at that time, which was a requirement of the United States Postal Service, where they wanted the handwritten digits on postal mail to be automatically sorted by an analysis of the digits on the postal.

Convolutional neural networks was one of the forerunners at that time of performance on the USPS of the MNIST data set. In the mid 90's also came the support vector machines SVMs, which was backed by an elegant theory and also started performing very well on these applications and for a large part between the mid 90's and perhaps the first decade of the 21st century machine learning applications were dominated by what today are known as traditional machine learning algorithms such as support vector machines, boosting, bagging and many other variants of these decision trees many other variants of these algorithms.

In the mid 2000's Geoffrey Hinton and Ruslan Salakhutdinov came up with a hierarchical feature learning algorithm also known as unsupervised retraining, using which they could initialize the weights for deep neural network and they showed that this could give distinct advantages of neural networks to outperform other methods, that was the seed of the deep learning revolution

in the years following there were further efforts to improve the training of deep neural networks, until then neural networks could not be trained with many layers which seem to be a major limitation.

This culminated in the development of the Alex Net developed by Alex Krizhevsky in 2012 to weight the image net challenge. The image net challenge is a data set consisting of about 1000 categories images consisting of objects of 1000 categories with over a million images at that time. And AlexNet outperformed all competitors by significant margin in 2012 taking the entire attention of the computer vision community.

Since then, every year's winner of the image net challenge has been a deep neural network, not only the image net challenge but even other benchmark challenges in vision have largely been dominated by deep neural networks and that has led to the golden age for deep neural networks, which we hope to see of the rest of this course.

(Refer Slide Time: 09:10)



**McCulloch-Pitts Neuron**


- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)
- $g$  aggregates the inputs and the function  $f$  takes a decision based on this aggregation
- The inputs can be excitatory or inhibitory
- $y = 0$  if any  $x_i$  is inhibitory, else

$$g(x_1, x_2, \dots, x_n) = g(\mathbf{x}) = \sum_{i=1}^n x_i$$

$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{if } g(\mathbf{x}) \geq \theta \\ 0 & \text{if } g(\mathbf{x}) < \theta \end{cases}$$

- $\theta$  is a thresholding parameter



Starting with the McCulloch-Pitts Neuron, McCulloch was a neuroscientist and Pitts was a logician proposed a simple computational model of the neuron in 1943 and the way this model work was you have a function  $g$  which aggregates a bunch of inputs which were all binary inputs at that time and a function  $f$  that took a decision based on the aggregation. So the inputs at that time were considered to be excitatory or 1 or inhibitory, a 0 or minus 1 depending on what notation you choose for binary numbers.

So, your  $g$  function was denoted as summation over each of your  $x_i$  and your final output  $y$  was given as  $f(g(x))$ , where the vector  $x$  denotes in different directions in the inputs in the input space which was given by 1, if  $g(x) > \theta$  and 0 if  $g(x) < \theta$ . This is what is illustrated here on the right, so you have a set of  $n$  inputs, they all coming into the function  $g$ , which aggregates them and  $f$  is the function that acts on  $g$  based on a pre-specified threshold  $\theta$  and that decides the final output, which also is a binary value.

(Refer Slide Time: 10:52)

**McCulloch-Pitts Neuron**

$y \in \{0, 1\}$      $y \in \{0, 1\}$      $y \in \{0, 1\}$      $y \in \{0, 1\}$      $y \in \{0, 1\}$

$x_1$   $x_2$   $x_3$      $x_1$   $x_2$   $x_3$      $x_1$   $x_2$   $x_3$      $x_1$   $x_2$      $x_1$

A McCulloch-Pitts Unit    AND function    OR function    NOR function    NOT function

- Feedforward MP networks can compute any Boolean function  $f: \{0, 1\}^n \rightarrow \{0, 1\}$
- Recursive MP networks can simulate any Deterministic Finite Automaton (DFA) (See this paper<sup>1</sup> for more information)

<sup>1</sup>Forcada, Neural Networks: Automata and Formal Models of Computation, 2002  
 Vineth N B. (IIT-H)    4.1 Neural Networks Review

Here are a few examples of how the McCulloch-Pitts model of the neuron can be used to simulate different logic gates. Here is the basic simple McCulloch-Pitts neuron, here is the McCulloch-Pitts neuron as an AND function, so you have  $x_1$ ,  $x_2$ ,  $x_3$  which are aggregated and the threshold which is chosen to be 3, so the  $\theta$  from the previous slide has been chosen to be 3 here which means this neuron McCulloch-Pitts neuron would give an output 1 only if all the 3 inputs were 1, because that is when it would exceed the threshold 3, a simple example of an AND gate.

Similarly, you could use a similar approach to simulate an OR function, whether threshold would be 1, if either of  $x_1$ ,  $x_2$  or  $x_3$  is a 1, this condition is threshold is satisfied and you get a 1 as the output and so the NOR function you have the McCulloch-Pitts neuron to be something like this, where if you notice there is a slight change in notation and these inputs now designate that these

are inhibitory inputs and not excitatory inputs, which means if one of them is on, it is definitely going to give an output as 0 and you can clearly see that as long as the threshold is greater than 0 here you would get an output.

Similarly, the NOT function is another setting where you have a single input and it is an inhibitory input and if you keep the threshold as 0, you can get your output to be 1 to simulate the NOT gate. As you can see, you can probably simulate most logic gate functions using the McCulloch-Pitts neurons, in fact it is possible to show that not a single McCulloch-Pitts neuron.

But the network of McCulloch- Pitts feed forward neurons which means you keep taking input and feed-forward that input to a set of McCulloch-Pitts neurons you can compute any Boolean function  $f$  that goes around Boolean input in  $n$  dimensions to an output that is Boolean in 1 dimension rather which means if you had from machine learning if you had an  $n$  dimensional input and the output is a binary classification problem, you could solve it as long as your inputs and outputs they are all binary.

Early work has shown that recursive McCulloch-Pitts networks can actually simulate any deterministic finite automata. We are not going to describe that here, but if you are interested you can look at this work to understand this connection between neural networks and automator in computer science.

(Refer Slide Time: 14:04)

### Perceptrons

- Frank Rosenblatt, an American psychologist, proposed the perceptron model (1958)
- Later refined and carefully analyzed by Minsky and Papert (1969)
- A more general computational model than McCulloch-Pitts neurons
- Inputs are no longer limited to boolean values

$$g(x_1, x_2, \dots, x_n) = g(\mathbf{x}) = \sum_{i=1}^n w_i * x_i$$

$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{if } g(\mathbf{x}) \geq \theta \\ 0 & \text{if } g(\mathbf{x}) < \theta \end{cases}$$

Vineeth N B (IIT-H)
§4.1 Neural Networks Review

As I just mentioned a few minutes ago Frank Rosenblatt was the psychologist who proposed the perceptron model in 1958, this was refined much later in the later part 60's very carefully analysed by Minsky and Papert who concluded that this cannot work in certain settings, but before we go there, let us introduce what a perceptron is. A perceptron was a generalization of the McCulloch-Pitts neuron and now the inputs were no longer limited to Boolean values, you could have any real value as input to a perceptron.

In addition, each of these inputs were weighted by a set of values which are denoted as  $w_1$  to  $w_n$  which means mathematically your perceptron is going to look like you have your  $g$  function, which was what we had with the McCulloch-Pitts neurons, which takes an input vector  $x$  of  $n$  dimensions, once again this could be any  $n$  dimensions in any data you could be looking at a patient record and you may want to say whether the patient is at risk for cancer, so these each of these attributes could be say the patient blood group the patients age any real value not just binary values anymore.

And the output of the  $g$  function is  $w_i x_i$  an inner product between a vector  $w$  and the input vectors  $x$  and the final output is again  $f(g(x))$ , which is going to be 1, if  $g(x)$  is greater than threshold or 0, if  $g(x)$  is less than a threshold, as you can see it is a generalization of the McCulloch-Pitts neuron where the inputs could be beyond and there are weights which multiply the inputs.

(Refer Slide Time: 16:22)

## Perceptrons

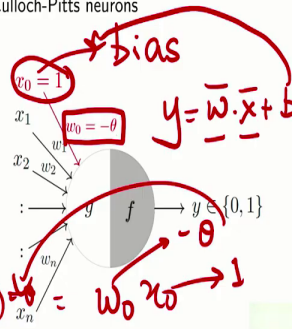
- Frank Rosenblatt, an American psychologist, proposed the perceptron model (1958)
- Later refined and carefully analyzed by Minsky and Papert (1969)
- A more general computational model than McCulloch-Pitts neurons
- Inputs are no longer limited to boolean values

- A more accepted convention

$$g(x_1, x_2, \dots, x_n) = g(\mathbf{x}) = \sum_{i=0}^n w_i * x_i$$

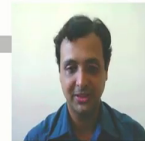
$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{if } g(\mathbf{x}) \geq \theta \\ 0 & \text{if } g(\mathbf{x}) < \theta \end{cases}$$

where  $x_0 = 1$  and  $w_0 = -\theta$



Vineth N B (IIT-H)

§4.1 Neural Networks Review



A more general way of writing a perceptron is where the index starts at 0 instead of 1. And the 0 is simply to substitute the threshold on as the part of the equation itself, rather you are now going to have  $i$  is equal to 0 to  $n$ , where  $x_0$  is going to be 1, it is a constant input 1 all of the time. Today this is also known as bias. So, if you look at a perceptron as modelling a line, remember the equation of a line can be written as  $w$  is equal to say  $w \cdot x + b$  where each of  $w$  and  $x$  can be a vector of a certain dimension say,  $b$  dimensions.

So,  $b$  is the weight  $w$  not that we are talking about which is multiplied by the value 1, that is now subsumed into a common submission which goes from  $i$  is equal to 0 to  $n$  where  $w_0$  is going to be denoted as  $-\theta$  and the  $x$  corresponding to  $w_0$  which is  $x_0$  will always be 1. Why do we do it? The reason we do it is now your output  $y$  which is  $f(g(x))$  will be 1 when  $g(x)$  is simply greater than or equal to  $\theta$  and 0 if  $g(x)$  is less than  $\theta$ .

Earlier we had instead 0, we had  $\theta$  which was the threshold, we are now subsuming that on the left hand side of that equation inside the  $g(x)$ , we are bringing the  $\theta$ , which means now  $g(x)$  will be the old  $g(x)$ , let us call that  $\hat{g}(x) - \theta$ , so that  $-\theta$  is now written as  $w_0$  which is  $-\theta x_0$ , so this is  $-\theta x_0$  is 1 which is what contributes to this  $-\theta$ . So, this is a more accepted convention of writing out a perceptron.

(Refer Slide Time: 18:41)



## Perceptron Learning Algorithm

### Algorithm 1 Perceptron Learning

```
w = [w0, w1, w2, ..., wn]  
x = [1, x1, x2, ..., xn]  
P ← input with labels 1;  
N ← input with labels 0;  
Initialize w randomly;  
while !convergence do
```



NPTEL



Vineeth N B (IIT-H)

§4.1 Neural Networks Review



Let us now look at how the weights of the perceptron are derived. So, you have a set of weights  $w_0$  to  $w_n$ , you have a set of inputs 1 and then  $x_1$  to  $x_n$ , remember 1 corresponds to the input for the weight  $w_0$ , you have an input with certain labels with label 1, let us call that set of inputs to be the positive class or P and you have another set of inputs with a label 0, which let us call the class as the negative class or the set N. We start by initializing the vector  $w$  randomly and then we loop over a set of steps, let us see what they are.

(Refer Slide Time: 19:31)

## Perceptron Learning Algorithm

### Algorithm 1 Perceptron Learning

```
w = [w0, w1, w2, ..., wn]  
x = [1, x1, x2, ..., xn]  
P ← input with labels 1;  
N ← input with labels 0;  
Initialize w randomly;  
while !convergence do  
    Pick random  $x \in P \cup N$ 
```

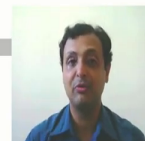


NPTEL



Vineeth N B (IIT-H)

§4.1 Neural Networks Review



The first step says, randomly pick a point and input data from  $P$  or  $N$ , rather you select  $x$  coming from  $P \cup N$ .

(Refer Slide Time: 19:44)



**Perceptron Learning Algorithm**

---

**Algorithm 1** Perceptron Learning

```
w = [w0, w1, w2, ..., wn]  
x = [1, x1, x2, ..., xn]  
P ← input with labels 1;  
N ← input with labels 0;  
Initialize w randomly;  
while !convergence do  
    Pick random x ∈ P ∪ N  
    if x ∈ P and ∑i=0n wi * xi < 0 then  
        |
```

Vineeth N B (IIT-H) §4.1 Neural Networks Review



If  $x$  belongs to  $P$ , which means it is a positive class and if your output was less than 0 remember for a past positive class you would have wanted your output to be greater than 0.

(Refer Slide Time: 20:00)



**Perceptron Learning Algorithm**

---

**Algorithm 1** Perceptron Learning

```
w = [w0, w1, w2, ..., wn]  
x = [1, x1, x2, ..., xn]  
P ← input with labels 1;  
N ← input with labels 0;  
Initialize w randomly;  
while !convergence do  
    Pick random x ∈ P ∪ N  
    if x ∈ P and ∑i=0n wi * xi < 0 then  
        | w = w + x  
    if x ∈ N and ∑i=0n wi * xi ≥ 0 then  
        |
```

Vineeth N B (IIT-H) §4.1 Neural Networks Review






But if it is less than 0 you simply add  $x$  to  $w$ , why is this so? We will see in a moment and if  $x$  belongs to the negative class and your output turned out to be positive or non-negative could be equal to 0.

(Refer Slide Time: 20:20)

**Perceptron Learning Algorithm**

```
Algorithm 1 Perceptron Learning
w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while /convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and ∑i=0n wi * xi < 0 then
    w = w + x
  if x ∈ N and ∑i=0n wi * xi ≥ 0 then
    w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */
```

Why would this work?



Vineeth N B (IIT-H) §4.1 Neural Networks Review

you once again subtract  $x$  from  $w$ , now you could ask the question, what about the scenario

where  $x$  belongs to  $P$  and  $\sum_{i=0}^n w_i x_i \geq 0$ , you simply do not do anything in that particular

scenario. Same case with  $x$  belonging to  $m$  and the summation being less than  $0$ , you assume things are good and do not worry about the weights in that scenario. Now, we have a termination criterion here, which says, until convergence keep doing this, so how do you converge?

You converge when all of your inputs are classified correctly by the  $w$ 's that you have. What does classified correctly mean? For all the data points  $P$  the output of a perceptron was greater than or equal to  $0$  and for all your inputs coming from the set  $N$  the output of the perceptron is less than  $0$ . That is the overall idea of the perceptron learning algorithm, but now let us ask the question. Why would this work? Why should we add  $x$  to  $w$ ? Why does that constitute a learning procedure and why would that improve the quality of  $w$ ? Let us see that a bit more carefully.

(Refer Slide Time: 21:42)

### Perceptron Learning Algorithm



---


**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and ∑i=0n wi * xi < 0 then
    | w = w + x
  if x ∈ N and ∑i=0n wi * xi ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */
        
```

- Why would this work?
- We are interested in finding the line  $\sum_{i=0}^n w_i * x_i = 0$  or  $w^T x = 0$  which divides the input space into two halves (binary classifier)



Remember as we said on the previous slide that a perceptron is a model of a line, because it is effectively  $w^T x + b$ , so it is the equation of the line and the lines equation is given by

$\sum_{i=0}^n w_i x_i = 0$ , this line divides your input space into two halves, the halves of inputs where

$\sum_{i=0}^n w_i x_i < 0$  and the halves of the inputs where  $\sum_{i=0}^n w_i x_i \geq 0$ .

(Refer Slide Time: 22:27)

### Perceptron Learning Algorithm



---


**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and ∑i=0n wi * xi < 0 then
    | w = w + x
  if x ∈ N and ∑i=0n wi * xi ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */
        
```

- Why would this work?
- We are interested in finding the line  $\sum_{i=0}^n w_i * x_i = 0$  or  $w^T x = 0$  which divides the input space into two halves (binary classifier)
- Every point (x) on this line satisfies the equation  $w^T x = 0 \Leftrightarrow \bar{w} \cdot \bar{x} = 0 \Leftrightarrow \sum_i w_i x_i = 0$



Every point on that line satisfies the equation  $w^T x = 0$ , remember we say  $w^T x = 0$ , this is equivalent to saying  $w \cdot x = 0$ , which is equivalent to  $\sum_{i=0}^n w_i x_i \geq 0$ . This is something that we keep interchangeably using for the rest of this course, please keep in mind that whether we say  $w^T x = 0$  or  $w \cdot x = 0$  or  $\sum_{i=0}^n w_i x_i \geq 0$  all of them are the same. We know now that every point  $x$  on that line has to satisfy the equation of that line, which means  $w^T x = 0$ .

(Refer Slide Time: 23:15)

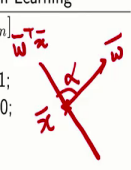
### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning


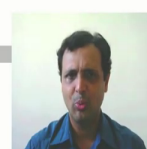
```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
    Pick random x ∈ P ∪ N
    if x ∈ P and ∑i=0n wi * xi < 0 then
        | w = w + x
    if x ∈ N and ∑i=0n wi * xi ≥ 0 then
        | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */
        
```



- Why would this work?
- We are interested in finding the line  $\sum_{i=0}^n w_i * x_i = 0$  or  $w^T x = 0$  which divides the input space into two halves (binary classifier)
- Every point (x) on this line satisfies the equation  $w^T x = 0$
- What can you tell about the angle ( $\alpha$ ) between  $w$  and any point (x) which lies on this line?
- The angle is  $90^\circ$  ( $\because \cos \alpha = \frac{w^T x}{\|w\| \|x\|}$ )
- Since the vector  $w$  is perpendicular to every point on the line it is actually perpendicular to the line itself

Vineth N B (IIT-H)
54.1 Neural Networks Review

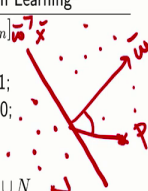
Now, what can you tell about the angle between  $w$  and any point  $x$  which lies on that line? This comes from geometry, the angle will always be 90 degrees, rather for the equation of any line, let us say you take the equation of any line which is given by  $w^T x$  remember once again that the intercept  $b$  is going to get subsumed into  $w_0$ , the vector  $w$  will always be perpendicular to the line. Why is that the case?

Because you have  $\cos \alpha$ , so if you take a point  $x$  here, let us take a particular point  $x$  on this line, the angle between  $x$  and  $w$  which is say given by  $\alpha$  is given by  $w^T x / \|w\| \|x\|$ , a simple expansion of the dot product. And angle we know has to be 0 because  $w^T x$  has to satisfy 0 for all points of the line and hence the angle has to be 90 degrees. Rather, the vector  $w$  is perpendicular to points on the line, why does this matter?




(Refer Slide Time: 24:41)

**Perceptron Learning Algorithm**

```
Algorithm 1 Perceptron Learning
w = [w_0, w_1, w_2, ..., w_n]
x = [1, x_1, x_2, ..., x_n]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and w^T x < 0 then
    w = w + x
  if x ∈ N and w^T x ≥ 0 then
    w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */
```



What will be the angle between vector  $x \in P$  and  $w$ ? Less than  $90^\circ$



Vineeth N B (IIT-H) §4.1 Neural Networks Review

Now, let us look at a point belonging to class P, remember this point does not lying on the line, it lies on this you wanted to lie on the side of the line, where  $w^T x \geq 0$ , because it belongs to the positive class, you want your perceptron to classify this point correctly, which means the perceptron should give you a value greater than or equal to 0. Let us see what it would be.

If you want the perceptron to classify this point correctly, the angle between the vector  $x$  to P and  $w$  has to be less than 90 degrees. Why so? Let us again write out that line, this is the line which is given by  $w^T x$ , remember this is  $w$ , this is the set P, this is the set N, so all the input points which have a negative class are one side of that line, all the input points which are the positive side of the line belong to the positive class. So, now any point in the positive class lies somewhere here, which means its angle is now going to be less than 90 degrees.



(Refer Slide Time: 26:06)

### Perceptron Learning Algorithm

---


**Algorithm 1** Perceptron Learning

```
w = [w0, w1, w2, ..., wn]  
x = [1, x1, x2, ..., xn]  
P ← input with labels 1;  
N ← input with labels 0;  
Initialize w randomly;  
while !convergence do  
  Pick random x ∈ P ∪ N  
  if x ∈ P and wTx < 0 then  
    | w = w + x  
  if x ∈ N and wTx ≥ 0 then  
    | w = w - x  
end  
/* algorithm converges when all the inputs  
are classified correctly */
```



- What will be the angle between vector  $x \in P$  and  $w$ ? Less than  $90^\circ$
- What will be the angle between vector  $x \in N$  and  $w$ ? Greater than  $90^\circ$
- Ponder and convince yourself this is the case
- For  $x \in P$  if  $w^T x < 0$  then it means that the angle ( $\alpha$ ) between this  $x$  and the current  $w$  is greater than  $90^\circ$

Vineth N B (IIT-H)§4.1 Neural Networks Review



Similarly, what about the angle between the angle  $x$ , belonging to a negative set and  $w$ , this should be straightforward, it has to be greater than  $90$  degrees for the same way we should visualize the positive class a minute ago. If you do not get it, you can spend a couple of minutes drawing it and you see that this should work.

Now, if you have point  $x$  belonging to  $P$  for which  $w^T x$  becomes less than  $0$ , it means that the angle  $\alpha$  between this  $x$  and the current  $w$  is greater than  $90$  degrees, which you do not want to. You want to change  $w$  to ensure this does not happen, but currently it is greater than  $90$  degrees.



(Refer Slide Time: 26:59)

### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and wTx < 0 then
    | w = w + x
  if x ∈ N and wTx ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */

```

- What will be the angle between vector  $x \in P$  and  $w$ ? Less than  $90^\circ$
- What will be the angle between vector  $x \in N$  and  $w$ ? Greater than  $90^\circ$
- Ponder and convince yourself this is the case
- For  $x \in P$  if  $w^T x < 0$  then it means that the angle ( $\alpha$ ) between this  $x$  and the current  $w$  is greater than  $90^\circ$
- But we want it to be less than  $90^\circ$
- How is adding  $x$  to  $w$  helping us?

Vineeth N B (IIT-H)
§4.1 Neural Networks Review

So, what do we do? We wanted to be less than 90 degrees. So, we are saying that we are going to add  $x$  to  $w$  to achieve this purpose and make the angle between  $w$  and this  $x$  belong to  $P$  less than 90 degrees.

(Refer Slide Time: 27:16)

### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and wTx < 0 then
    | w = w + x
  if x ∈ N and wTx ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */

```

- What happens to the new angle ( $\alpha_{new}$ ) when  $w_{new} = w + x$

$$\cos \alpha_{new} \propto w_{new}^T x$$

Vineeth N B (IIT-H)
§4.1 Neural Networks Review

Why do you think that helps? Let us understand that. Let us consider this  $w_{new}$  to be  $w + x$ , which means we have added that  $x$  which belongs to  $P$  to  $w$  and got the new  $w$  in the iteration, let us try to understand what the new angle will be.

(Refer Slide Time: 27:35)

### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and wTx < 0 then
    | w = w + x
  if x ∈ N and wTx ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */

```

- What will be the angle between vector  $x \in P$  and  $w$ ? Less than  $90^\circ$
- What will be the angle between vector  $x \in N$  and  $w$ ? Greater than  $90^\circ$
- Ponder and convince yourself this is the case
- For  $x \in P$  if  $w^T x < 0$  then it means that the angle ( $\alpha$ ) between this  $x$  and the current  $w$  is greater than  $90^\circ$
- But we want it to be less than  $90^\circ$
- How is adding  $x$  to  $w$  helping us?

Vineth N B (IIT-H)
§4.1 Neural Networks Review

We just found out that when  $w^T x < 0$  for a point belonging to P the angle turns out to be greater than  $90^\circ$  which we do not want we wanted to be less than  $90^\circ$ .

(Refer Slide Time: 27:47)

### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and wTx < 0 then
    | w = w + x
  if x ∈ N and wTx ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */

```

- What happens to the new angle ( $\alpha_{new}$ ) when  $w_{new} = w + x$

$\cos \alpha_{new} \propto w_{new}^T x$

Vineth N B (IIT-H)
§4.1 Neural Networks Review

Now, Let us see what happens for  $w_{new}$ . We know that  $\cos \alpha_{new} = \frac{w_{new}^T x}{\|w_{new}\| \|x\|}$ , so we are going to ignore the denominator and simply say  $\cos \alpha_{new} \propto w_{new}^T x$

(Refer Slide Time: 28:06)

### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning



```


w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
    Pick random x ∈ P ∪ N
    if x ∈ P and wTx < 0 then
        w = w + x
    if x ∈ N and wTx ≥ 0 then
        w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */
        
```

What happens to the new angle ( $\alpha_{new}$ ) when  $w_{new} = w + x$

$$\cos \alpha_{new} \propto w_{new}^T x$$

$$\propto (w + x)^T x$$

Vineeth N B (IIT-H)
§4.1 Neural Networks Review


### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
    Pick random x ∈ P ∪ N
    if x ∈ P and wTx < 0 then
        w = w + x
    if x ∈ N and wTx ≥ 0 then
        w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */
        
```



What happens to the new angle ( $\alpha_{new}$ ) when  $w_{new} = w + x$


$$\cos \alpha_{new} \propto w_{new}^T x$$

$$\propto (w + x)^T x$$

$$\propto w^T x + x^T x$$

$$\propto \cos \alpha + x^T x$$

Vineeth N B (IIT-H)
§4.1 Neural Networks Review


Which can be set is proportional to  $(w + x)^T x$  because  $w_{new} = w + x$ , which can be written as  $w^T x + x^T x$  which can be written as  $\cos \alpha$ , because we know that  $w^T x \propto \cos \alpha + x^T x$ .

(Refer Slide Time: 28:30)

### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning

```



w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and wTx < 0 then
    | w = w + x
  if x ∈ N and wTx ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */


```

What happens to the new angle ( $\alpha_{new}$ ) when  $w_{new} = w + x$

$$\begin{aligned} \cos \alpha_{new} &\propto w_{new}^T x \\ &\propto (w + x)^T x \\ &\propto w^T x + x^T x \\ &\propto \cos \alpha + x^T x \end{aligned}$$

$\cos \alpha_{new} > \cos \alpha$



Which means because  $x^T x$  has to be a positive quantity because it is a dot product of a vector with itself so all values will get squared and you will add up all of those values. You know that  $\cos \alpha_{new} > \cos \alpha$ .

(Refer Slide Time: 28:52)

### Perceptron Learning Algorithm

---

**Algorithm 1** Perceptron Learning

```

w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and wTx < 0 then
    | w = w + x
  if x ∈ N and wTx ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
are classified correctly */



```


What happens to the new angle ( $\alpha_{new}$ ) when  $w_{new} = w + x$

$$\begin{aligned} \cos \alpha_{new} &\propto w_{new}^T x \\ &\propto (w + x)^T x \\ &\propto w^T x + x^T x \\ &\propto \cos \alpha + x^T x \end{aligned}$$

$\cos \alpha_{new} > \cos \alpha$

- Thus  $\alpha_{new}$  will be less than  $\alpha$  and this is exactly what we want
- We can work out the math for the other case, when  $x \in N$  and  $w^T x \geq 0$  quite similarly



Rather, this tells us that  $\alpha_{new} < \alpha$  and hence we will get the right weight vector which will ensure that points belonging to P have the output of the perceptron to be greater than or equal to C. But you can work out a similar scenario for points belong to the negative class and you will

get a negative class where the output of the perceptron is greater than equal or 0, you can work out a very similar scenario, the only thing is you would a negative sign here and you will find that the new angle will be greater than the old angle which is what we would want for a negative class.

(Refer Slide Time: 29:33)

### Perceptron Learning Algorithm

```

Algorithm 1 Perceptron Learning
w = [w0, w1, w2, ..., wn]
x = [1, x1, x2, ..., xn]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
  Pick random x ∈ P ∪ N
  if x ∈ P and wTx < 0 then
    | w = w + x
  if x ∈ N and wTx ≥ 0 then
    | w = w - x
end
/* algorithm converges when all the inputs
   are classified correctly */

```

- What happens to the new angle ( $\alpha_{new}$ ) when  $w_{new} = w + x$ 

$$\cos \alpha_{new} \propto w_{new}^T x$$

$$\propto (w + x)^T x$$




$$\propto w^T x + x^T x$$

$$\propto \cos \alpha + x^T x$$

$$\cos \alpha_{new} > \cos \alpha$$
- Thus  $\alpha_{new}$  will be less than  $\alpha$  and this is exactly what we want
- We can work out the math for the other case, when  $x \in N$  and  $w^T x \geq 0$  quite similarly
- For a formal convergence proof, please see [this link](#)

Vineth N B. (IIT-H)

§4.1 Neural Networks Review

That should convince you that this perceptron learning algorithm will keep improving in each iteration. Now, when would it converge? We only know that it would keep improving in each iteration, but whether it will get us to that final solution?

We still have not proved it and probably not going to work out that detail here, but if you want a formal convergence prove to show that as the number of data points increase as the number of iterations increase you are going to get a solution as long as you can separate points originally into two classes P and N using a line, you can see this link shared here for the formal conversion is proof.