

**Applied Natural Language Processing**  
**Prof. Ramaseshan Ramachandran**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 33**  
**Perceptron**

(Refer Slide Time: 00:15)

**LAWS OF ASSOCIATION**

Aristotle's attempts on fundamental laws of learning and memory

$U\Sigma V^T = LSI$

<p><b>The law of similarity</b></p> <p>If two things are similar, the thought of one will tend to trigger the thought of the other - word2vec</p> <p>If you recollect one birthday, you may find yourself thinking about others as well</p>	<p><b>The law of contrast</b></p> <p>Seeing or recalling something may also trigger the recollection of something completely opposite</p>
<p><b>The law of contiguity</b></p> <p>Things or events that occur close to each other in space or time tend to get linked together in the mind</p> <p>If you found a snake in the corner of the street, every time you cross the corner, you tend to look for one. Events are conditioned based on the time and space</p>	<p><b>The law of frequency</b></p> <p>The more often two things or events are linked, the more powerful will be that association - think of next word prediction - strength of the association decides who is the probable candidate</p>

19 / 63

Now, a little bit about some of the associative laws which are very useful in the design of neural networks. one is the law of similarity. these laws are fundamental laws attempted by Aristotle, it is very old; in terms of learning and memory. There are 4 fundamental laws, one is the law of similarity, the second one is the law of contrast, third is contiguity and then fourth is the fluency. we also have to relate this to what we have in the natural language processing, then only they are useful otherwise you know it is set another individual set of laws, ok.

If two things are similar the thought of one will tend to trigger the thought of the other, ok. This is something that I am going to be talking about in future word2vec, but let us talk about the LSI that we have already learned. what does LSI do? Using the SVD it is able to really decompose the term-document matrix into 3 matrices, right. one is the left singular matrix we called it U and then we have a diagonal matrix where the elements are in the descending order it is the singular matrix and then another one which is of V, V this is the right singular matrix.

And we mentioned that the elements or the rows of  $U$  represent the words, right. it is equal to the size of the terms that we had. For example, if the term for the wanted space  $I$  will use this, this is your document, right. the column tells you various features of your documents and then rows give you the features of your words. And then we mentioned that after transforming the original matrix into this latent domain  $U$  the rows of  $U$  represent the word vector. It is not the representation of the term directly, but it represents a lot of similar words as part of that that is why we call that the word embedding.

So, in the same fashion the loss of similarity when we use one of these techniques either LSI or word2vec, we are able to capture a similar pattern. if you take one of the vectors, it should be able to tell you what are all other similar words that you have, related to the word that you have picked up. we will show one example later during the time this will become very clear, ok. in this case if you recollect or if you pick one word, the context related to the word is learned using the LSI as well as in the word vector, which will be talking about later.

So, it will be able to tell you what are all the related words with respect to some cosine distance; this is very useful in the natural language processing where we can pick up the similar words with respect to its context. In another example, if you recollect the birthday of your friend in the class again, you are celebrating it. immediately the thought will come, whose birthday is next right? that is something we can roughly relate to the law of similarity.

And then the law of contrast; seeing or recalling something may also trigger the recollection of something which is completely opposite, ok. If somehow you know we are able to find instead of this similarity the antonyms of each word. For example, I have trained the network in such a way that when you give one word it will always give me the opposite of those words or the words that are opposite to that, ok. that could be the law of contrast.

Law of contiguity. Things are even that occur close to each other in space or time tend to get linked together in the mind, ok. This funny example that I have quoted here, but I think it is true in many cases. This is one simple example. while walking back to your house when you find a snake in one corner of the street today, right and then if you are

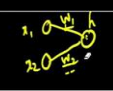
going to the same place every day from today onwards you always tend to look into that corner to see if you can find a snake again, ok. this you will call as the law of contiguity.

And then the law of frequency is something we spoke about even in the prediction of the next word, right. In if certain sets of words are connected through the context and its frequency is pretty high when you provide the first word it is able to tell you what could be the next word based on the frequency of the occurrences of those two words in the corpus. again the law of frequency is very useful in this case. if you look at this these are all patterns that we can use as part of the input to the system, right.

So, we should be able to capture you know for example, the similarity words in the case of word2vec that is how we really train the network. we have a set of context, and then I want to find out the middle word given the surrounding word, I would use the laws of similarity in that the patterns are found using the laws of similarity, ok. The prediction of the next word or the probability of a given sentence could also be found using the law of frequency, ok. there is some association even though it is pretty old. We are able to map these patterns into or these laws into our existing problem in hand, ok.

(Refer Slide Time: 07:51)

PERCEPTRON ✓



Neuron	Perceptron
Biological	A mathematical model of a biological neuron
Dendrites receive electrical signals	Perceptron receives mathematical values as input
Electro-chemical signals between Dendrites and axons	The weighted sum represents the total strength of the signal
The electro-chemical signals are not static	Weights change during the training process

20 / 63

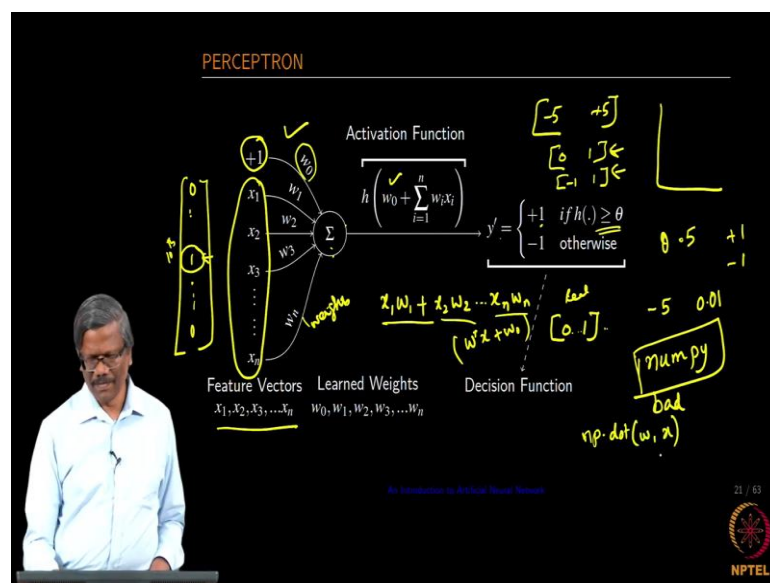
NPTEL

So, what are the differences between the perceptrons? We are going to be calling the artificial neuron as the perceptron. It is biological and perceptron is a mathematical model. Here dendrites receive the electrical signal perceptrons receive the values as input, electrochemical signals between the dendrites and the axon. whereas we are going

to have again there are certain weights for the example in the case of the perceptron, let us say this is your neuron we are going to be connecting with respect to some weights, ok.

So, the strength of the signal is represented by these weights in the same way we had seen in the biological neuron. The electrochemical signals are not static; you know we keep learning, so they are not very static. Even in this case during the training process these weights keep changing. Once the training is complete the weights are static, ok.

(Refer Slide Time: 09:11)



So, I am going to give you a very high-level overview of what a perceptron is and what we can do with that you know using this simple architecture. in this case, we have a set of inputs or we can call it as the feature vectors whose sizes are  $h \times 1$ , to  $x \times n$ , ok. And then we have a bias whose value is plus 1, which is connecting to the perceptron using the weights  $w_0$ . all the input feature vectors are connected to this element through the weights, ok. we call them as weights. the input is connected to the neuron through the weights.

So, for the sake of explanation I have actually divided this into two spaces, one is the activation functions. Once the values are received, they are summed in this neuron they are linearly summed and then there are some values that we have obtained, ok. for example, the value could be between minus 5 to plus 5, depending on the weights that we have used as the weight that are connecting the input and the neuron. we want to

actually translate the values between 0 and 1 or minus 1 and 1 or through some probability distribution where the values when they are summed would be equal to 1. there are so many ways we can do that.

So, the activation function that it does is it smashes the values of this into a point in this space. See for example, if I have minus 5 and then the value becomes using the activation function, it gives you 0.01 and my threshold let us say my threshold is 0.5. If the threshold is greater than 0.5 then the value is equal to plus 1 otherwise it is going to be minus 1 as given by this decision function. the activation function squashes the value between 0 and 1 and the decision function actually looks at the values squashed by the activation function. And then makes a decision whether to provide plus 1 to it or minus 1 to it depending on the threshold that we have set up.

So, we have the feature vectors coming into the neuron and then the weights are connecting the input and the neuron. There is an activation function that smashes the value between two values and there is a decision function that translates the activation function value into two different values which we can use to take the case of the sentiment analysis.

Assume that we are able to provide the input to the perceptron using a one-hot vector. Remember the one-hot vector and index of that particular element in that space will give you the word. For example, if it is the tenth element and my positive value is good. it will represent well. in this fashion I can keep feeding the feature vectors as input and the result would be expected as either positive or negative. for this I expect the perceptron to output as plus 1, and then for bad I expect the perceptron to output as minus 1. And then the activation function smashes the value between 0 and 1, and then the decision function decides whether it is positive or negative depending on the value it received from the activation function, ok. this is a very simple network that you have for perceptron.

Again the mathematical operations that we are going to be performing here are very simple. the one is we have a bias that is connecting to it. this sum that you are looking at here would be  $x_1 w_1$  plus  $x_2 w_2$  to  $x_n w_n$  or it is written in a very simple notation. It is a sum of  $I$  equal to 1 to  $n$ ,  $w_i x_i$ .

So, this is a simple dot product that you have. And then you finally, add that with the bias and then give it to the activation function. We will talk about what that activation

function is. When you provide the value completed to the activation function it maps it between two values. Let us say it is going to be mapping at between two values and these are all real values, and then based on what the values is the decision function either outputs plus 1 or minus 1 depending on what value this activation function has provided, ok.

So, it is a very simple mathematical model and I want you to go and then take a look at NumPy. a lot of linear vector operations are or rather linear algebraic operations are performed in this library NumPy and you can very easily translate this into NumPy and do it, instead of you know writing loops of instructions to add this you can just use one single instruction to save np dot like this, ok. this is clear now.

The input could be anything, right. It could be any real values or it could be a one-hot vector in the case of natural language processing. The output is going to be only two values, either plus 1 or minus 1 depending on what activation function these squashes, ok. it is a very simple example of a perceptron where there is only one element and this is good enough to linearly classify a set of vectors into two reasons, ok.

(Refer Slide Time: 17:05)

**PERCEPTRON LEARNING**

- ▶ Perceptron learns the weights ✓
- ▶ They are adjusted until the output is consistent with the target output in the training examples
- ▶  $w_j^{(k+1)} \propto (y - \hat{y})$
- ▶ The weights are updated as below  

$$w_j^{(k+1)} = w_j^{(k)} - \eta(y - \hat{y})x_{ij}$$
 where  $w_j^{(k)}$  is the weight parameter associated with the  $i^{th}$  input at  $k^{th}$  iteration

$\eta$  is the learning parameter and  $x_{ij}$  is the  $j^{th}$  attribute of the  $i^{th}$  training sample

- ▶ If  $(y - \hat{y}) = 0$ , no prediction error
- ▶ During the training the weights contributing most to the error require adjustments

$\eta = 0.01$       $\eta = 1.2$

22 / 63  
NPTEL

So, how do we teach the perceptron, or how does perceptron learn the weights? very simple. You know it is going to learn the weight, the weights are going to be your model. the first statement is it learns the weights. Let us see how it will learn the weights. They are adjusted until the output is consistent with the target output in the training examples.

you keep changing the weights in such a way that the estimated output is very close to the target output. Since we know the relationship between the input vectors and the output vectors, we are estimating the model and the model parameters are here  $w$  and  $w_{naught}$ ,  $w$  is the weight connecting the input and the perceptron and  $w_{naught}$  is the weight connecting the bias and the perceptron, ok.

So, if we have iteration going from 1 to  $k$ , right. we need to keep eyes identifying the weight that you want to update. And this weight is proportional to, the new weight that you are going to be using to updates the weight connecting the input, and the perceptron is proportional to the error that is computed. This is your target and this is your estimate. intuitively you can very easily say that the next weight is given by or the new weight is given by the old weight, and there is one new parameter that I have here and here we have the error and the input that is connecting, ok.

The parameter that we have here is called the learning parameter. The  $\eta$  is the learning parameter, and it is adjusted to actually help you descend. If the  $\eta$  value is very high the learning jumps in this fashion, if  $\eta$  is very small it slowly and steadily reaches this. Normally, we do some kind of estimation for this you know by looking at the training samples and then see how the weights are adjusted and how the errors are jumping from one point to the other during the iteration. We keep adjusting this, ok.

So, we look at the way the training goes and adjust this and then finally, settle down to some number, so until then we empirically change these values, ok. the  $\eta$  parameter is updated based on the experience that you gain in the model estimation. normally it ranges from 0.1 to 0.01, ok, rarely it goes below this point. And you cannot just have  $\eta$  equal to 1, 2 and so on. It will be very drastic for example, if you have the error surface in this fashion and if you have a big  $\eta$  it might go wild in this fashion. It is very difficult to control that. ideally it should descend in this fashion, ok. it is very crucial to really set the right parameter for the  $\eta$ .

(Refer Slide Time: 21:21)

**PERCEPTRON LEARNING**

- ▶ Perceptron learns the weights ✓
- ▶ They are adjusted until the output is consistent with the target output in the training examples
- ▶  $w_j^{(k+1)} \propto (y_i - \hat{y}_i)$
- ▶ The weights are updated as below  
$$w_j^{(k+1)} = w_j^{(k)} - \eta (y_i - \hat{y}_i) x_{ij}$$
where  $w_j^{(k)}$  is the weight parameter associated with the  $i^{\text{th}}$  input at  $k^{\text{th}}$

iteration

$\eta$  is the learning parameter and  $x_{ij}$  is the  $j^{\text{th}}$  attribute of the  $i^{\text{th}}$  training sample

- ▶ If  $(y_i - \hat{y}_i) \approx 0$ , no prediction error
- ▶ During the training the weights contributing most to the error require adjustments

$e = 2$     $w \rightarrow$

Targ.  $y = 1$   
est.  $\hat{y} = -1$

22 / 63  
NPTEL

So, when  $y$  minus  $\hat{y}$  equal to 0; that means, there is no I would say approximately equal to 0. There is no prediction error that is when you know we normally set this to 1 into  $e$  for minus 5, it is a very small value. during the training the weights contributing most to the error require adjustments, ok.

I am going to ask you to do one exercise in this, where if supposing if my output or the target output is 1 and the estimated output is minus 1, ok. this is the estimated one, this is the target value, ok. when you have this set, so how will you update  $w$ ? what is the kind of adjustment you make to  $w$  so that it becomes closer to 1? right now you know it is minus 1. when you do this operation the error equals 2. in which direction you update the weights? Ok. This is the question for you. I like you to go and then refer to some books and figure out what could be the adjustment that you want to make. you know that we need to make some adjustments, but what kind of adjustment.



(Refer Slide Time: 23:19)

**PERCEPTRON LEARNING**

- ▶ Perceptron learns the weights ✓
- ▶ They are adjusted until the output is consistent with the target output in the training examples
- ▶  $w_j^{(k+1)} \propto (y - \hat{y})$
- ▶ The weights are updated as below
 
$$w_j^{(k+1)} = w_j^{(k)} - \eta (y - \hat{y}) x_{ij}$$
 where  $w_j^{(k)}$  is the weight parameter associated with the  $i^{th}$  input at  $k^{th}$  iteration

iteration

$\eta$  is the learning parameter and  $x_{ij}$  is the  $j^{th}$  attribute of the  $i^{th}$  training sample

If  $(y - \hat{y}) \approx 0$ , no prediction error

During the training the weights contributing most to the error require adjustments

$1.e-5$

Target  $y = -1$   
 $\hat{y} = 1$

Target  $y = 1$   
 $\hat{y} = -1$   
 est.  $\hat{y} = -1$

22 / 63  
 NPTEL

The same example I am sorry another example would be the target is minus 1 and y hat is 1, in which way you would adjust the weights? Either increase the weights or decrease the weights by certain fashion? We know that it is proportional to the error, right; in which way you would move the weights, ok.

(Refer Slide Time: 23:49)

**ALGORITHM FOR PERCEPTRON LEARNING**

- 1: Total number of input vectors =  $k$  ✓
- 2: Total number of features =  $n$  ✓
- 3: Learning parameter  $\eta = 0.01$ , where  $0 < \eta < 1$
- 4: epoch<sup>1</sup> count  $t = 1, j = 1$
- 5: Initialize weights  $w_j$  with random numbers
- 6: Initialize the input layer with  $\vec{x}_j$
- 7: Calculate the output using  $\sum w_j x_j + w_0$
- 8: Calculate the error  $(y - \hat{y})$ .
- 9: Update the weights  $w_j(t+1) = w_j - \eta (y - \hat{y}) x_j$
- 10: Repeat steps 7 and 9 until the error is less than  $\theta$  or a predetermined number of epochs have been completed.

To provide a stable weight update for this step,  $w_j(t+1) = w_j - \eta (y - \hat{y}) x_j$ , we require a small  $\eta$ . This results in slow learning. Bigger  $\eta$  would be good for fast learning. What are the problems? . What is the compromise?

<sup>1</sup>An epoch is one complete presentation of the data set to be learned to a learning machine.

(1-hot vector)

600

23 / 63  
 NPTEL

So, this is a very simple algorithm for the perceptron. Let us assume that we have about  $k$  vectors and then there are  $n$  number of features that we have for each vector and then we

have set the learning parameter to be 0.01 and then epoch count is t equal to 1 and j equal 1.

So, I have used a new word epoch here. The epoch is one complete presentation of the data set. For example, we have a let us say we are going to be doing the sentiment analysis where we have words representing the one-hot vector, right. we have about 500 words that we want this system to learn either to be as positive words or negative words. I am providing a one-hot vector, ok, so about 5-600 of them. they represent one there. Take every word, compute the error, make the change until it satisfies the input-output relationship, and then take the next word, and keep going. that is one way.

(Refer Slide Time: 25:39)

ALGORITHM FOR PERCEPTRON LEARNING

Epoch	$x_1$	$x_2$	b	$w_0$	$w_1$	$w_2$	h	y	Target	Target-y	DeltaW	$w_0^{NEW}$	$w_1^{NEW}$	$w_2^{NEW}$
1	0	0	-1	--	--	--	--	--	--	0	0	0	0	0
	0	1	-1	0	0	0				1	1	1	0	1
	1	0	-1	1	0	1				1	1	0	1	1
	1	1	-1	0	1	1				1	0	1	1	0
2	0	0	-1	1	1	0				0	0	0	1	0
	0	1	-1	0	1	0				1	1	0	1	1
	1	0	-1	0	1	1				1	0	1	1	0
	1	1	-1	1	1	0				1	1	1	1	1
3	0	0	-1	1	1	1						0	0	1
	0	1	-1	0	0	1				The numbers are NOT computed only to demonstrate how updates occur during every epoch				
	1	0	-1	1	1	0						1	1	1
	1	1	-1	1	1	1						1	1	1

NPTEL

In this case, I am presenting the entire 600 one hot vectors in one go, not in one go one after the other, and then start making the adjustment every time, ok. this is one epoch. This is one way of presenting the data set.

We initialized the weights with random numbers, initialized the input layer with the first data, and calculate the out net output using this relationship. Calculate the error, update the weight based on this equation, ok, then repeat steps 1 through 9 or rather 7 through 9, till the error is less than the given threshold or a predetermined number of epochs are completed, ok. these are the 10 steps that you have to make the perceptron learn.

So, again this is for you to find out what is the optimal eta size, should be should it be big or small or very small or very bigok.

(Refer Slide Time: 27:35)

The slide is titled "ACTIVATION FUNCTIONS" in orange text. On the left, a man in a light blue shirt is speaking. To his right is a list of activation functions, each with a checkmark:

- ▶ Hard threshold ✓
- ▶ Sigmoid ✓
- ▶ Tanh ✓
- ▶ ReLu - Rectified Linear Unit ✓
- ▶ Leaky ReLu ✓
- ▶ Softmax ✓

To the right of the list is a hand-drawn diagram of a neural network with three layers of nodes. The nodes are connected by lines. The diagram is annotated with yellow text: "I/O" at the bottom left, "Hidden" in the middle, and "Softmax" at the bottom right. Above the diagram, there are three rows of handwritten vectors:  $[0 \ 1]$ ,  $[-1 \ 1]$ , and  $[ \dots ]$ . In the bottom right corner, there is a small red circular logo with the text "NPTEL" and "25 / 63" above it.

So, in this session, we are going to be talking about the activation function. We spoke about the activation function in the perceptron case, where we mentioned that it will output a value between 0, 1, 1, or minus 1 to 1 or it gives you a probability distribution where the sum of each element would be equal to 1, right.

So, let us see each one of them. There are several types of activation functions, one is the Hard threshold, the second one is a Sigmoid, third is the Hyperbolic Tanh function and then fourth one is the ReLu - Rectified Linear Unit and then there is another one called Leaky ReLu and the popular one in the natural language processing word is the Softmax which is usually used at the output layer. And the Sigmoid and Tanh usually used in the middle layers, ok.

So, there is network architecture where there would be more than one neural element, ok. you will have input connecting to neurons in this fashion and you will also have multi-class output in this fashion, ok. in most cases of the natural language processing you will have a minimum of 3, let us say 3 layers. We call this as the input layer, a hidden layer, and the output layer. In many neural networks you will find that the hidden layers are squashed using either Sigmoid or Tanh, and then the output is squashed by a Softmax.