in this lecture we're going to talk about generalizing the notions in logistic regression to multiple classes and a particular function that's seen wide use in multi-class classification and neural nets called soft max now we saw in logistic regression how a single score computed with a weight vector dot feature vector could convert into a probability that the data item being classified is in the class we want to generalize this where the label is not just plus one or minus one but y is in a one of many classes say cat dog table car different classes that we might want to classify an image into now for this we're now going to have multiple scores a score for each class and thus we need multiple weight vectors a weight vector for each class i'll just denote them this way these are each weight vectors just like the weight vectors we've seen and we'll still have a feature vector same dimensionality as the weight vectors each weight vector so each weight vector is d dimensions and so is the future vector and so we will get multiple scores i guess i've been writing w dot v rather than p dot w um it doesn't matter something you should appreciate that you're going to get the same value when you dot these two together whichever order you write them in so i will call this s super 1 to s super k but remember this isn't a new thing it's just a name for the familiar thing the score from that weight vector and that feature vector we just have k of them now one for each class so we've one input x but k scores you can sort of think of it as k different class virus competing to say no it's a cat no it's a dog right and each one produces a score but we don't want a score what we really want is the right answer which one is it but that's not a smooth notion and it also doesn't reflect our uncertainty right if i show you a bunch an image and ask you is it a cat a dog a plane a bird and you look at it and you're not sure but you think it's probably a dog that's different than being certain it's a dog and so what we really want is a smooth notion that reflects your uncertainty in other words we want we want the probability that y equals class i given x that's what we really want out of these scores for each i so we would like to be computing from score one a probability that y is 1 and from score k a probability that y is k and of course these probabilities must sum to 1. that's the most obvious problem we have some other obvious problems here these numbers can even be negative they're arbitrary real numbers right these scores can be negative and so we need a function that's going to take scores that are real numbers convert them to probabilities and and so that function is we call it soft max because it's in some sense trying to figure out what's the biggest score and identify that one but it's soft and that it converts to probabilities so there's a smooth gradient um and so the first thing it's going to do is get these onto a positive scale by exponentiating each of them take e to the score 1 e to the square k now that converts our real number scale to a strictly positive value which is closer to what we want because we want probabilities they're always positive it also magnifies the differences which is going to be good for focusing on the max right we want to focus when we want the max to stand out as having the highest probability it's a soft version of picking the max but we still want the max to get a lot of probabilistic weight so that's two reasons for exponentiating another reason is if you go if you work like we did in the logistic regression lecture you can ground some of this in log likelihoods and if you have log likelihood and you want to get it to probability right we're trying to get to probability then you want to exponentiate it first to get the log out so anyways we're going to for various reasons that i'm only giving you an intuitive handle on we're going to exponentiate these so so we're next going to convert these to e to the s1 to e to the sk right that's score one to score k so that's a simple conversion enough right and now now we want to normalize to one so we end up with e to the score from classifier 1 over some normalization z and likewise all the way down to e to the k score over z here z equals the sum over i equals 1 to k of e to the s basically the exponential of the score so we're basically summing the numerators to get z so that's just straightforward normalize this so it sums to one and so this leads us to this probability probability that y equals i given x equals e to the heighth wave weight vector dotted with v over z where z is just that same thing but summed for all the different eyes don't much like that box i'm going to fix it so what remains is to define a loss function for training to make this value that this computes right this isn't automatically the probability we're going to say it is but we need to define a loss function that will coerce it to be that and pretty much analogously without a lot of theoretical exploration for this class we're going to use the entropy the cross entropy again to force this score to this normalized score to be the probability so our loss function for a given data point x i that should be labeled y i remember that's a number between 1 and k and all of our weights i almost want to put two vectors it's a vector of vectors of weights now don't let that stress you um but it's going to be negative natural log the probability that y equals y i given x so notice that there's loss is computed only from the network the part of the network that has to do with the label y i like if this is class three we experience loss in class three for not saying one if this probability is one there won't be any loss but if this probability is less than one we'll have some loss in class three we don't experience any loss for having given probability in the other classes because that would be a sort of double penalizing right every bit of not saying one for class three is distributed somewhere else to another class but we're not penalizing over there so if we say if we say it's got a 0.8 probability of being a cat and it is supposed to be labeled a cat we'll get a loss for not having said one but the remaining point two will be distributed in the other classes and we will not get any loss added for that this so this loss um only causes us to to try to say a higher probability a higher score for category i for data point x i and in particular the gradient is of this loss function is going to look at the weights for the class it was supposed to be labeled as and increase those weights proportional to fee fee of x i whatever features x i is strong in we're going to try to increase those weights i know if it has any negative features we're going to decrease the corresponding weights in the ith part of the network we won't do anything based on this example won't do anything for the the parts of the network labeled with with superscripts that are not equal to the y i i'm sorry i'm using i for two meanings here when i say the i part of the network i mean the y is part of the network right y i is some 1 through k value and if it's three that's the network for labeling class three and that's the part that will get the weights adjusted from this training example okay so otherwise we just plug in our stochastic gradient descent with this loss function which is cross entropy loss and this is cross entropy loss based upon soft max for multiple classes this is soft max it's really soft hard max but nobody calls it that all right and then we can use stochastic gradient descent to train the weights and this can perform very well