使用梯度下降法的条件：

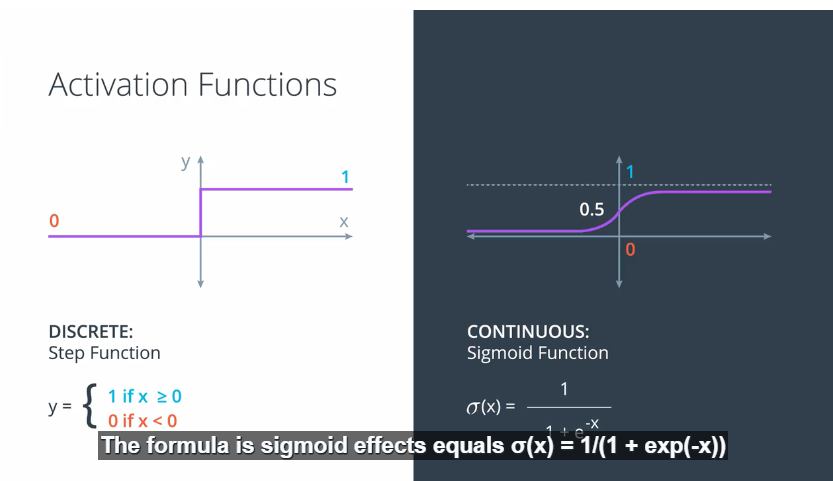
The error function should be differentiable

误差函数必须是可微的

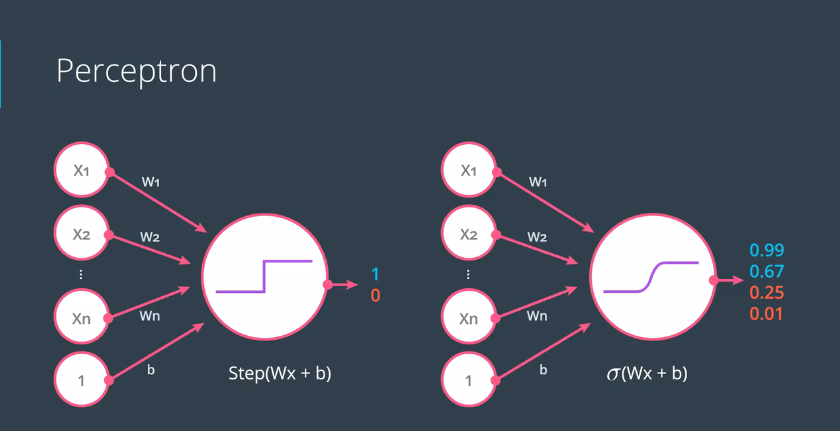
The error function should be continuous

误差函数必须是连续的；

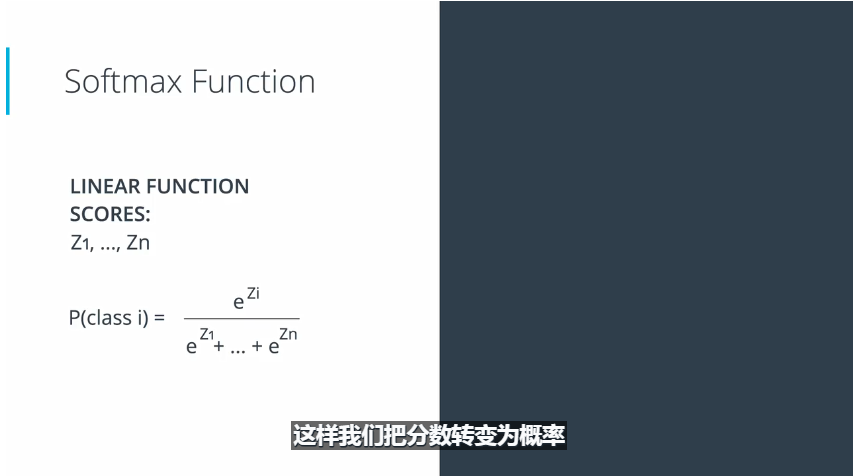
如何从离散函数变为连续函数：



比如在二维，二分类坐标系中距离目标直线越远的点，其正确分类概率越高，距离越大对应概率更高，这样就转换为连续的概率问题了；



Softmax函数



**Maximum Likelihood**

Probability will be one of our best friends as we go through Deep Learning. In this lesson, we'll see how we can use probability to evaluate (and improve!) our models.

更准确的模型，提供更高的概率(正确分类的概率)

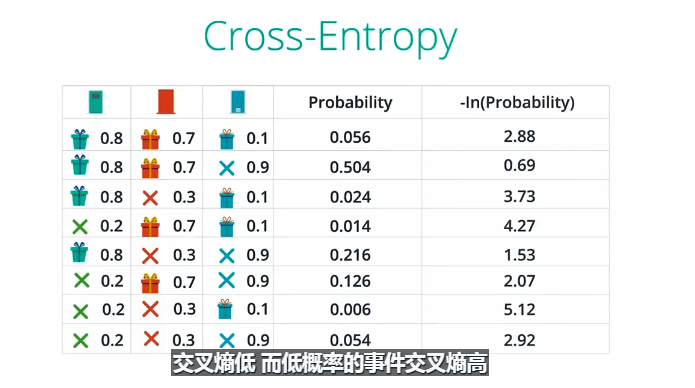
# Maximizing Probabilities

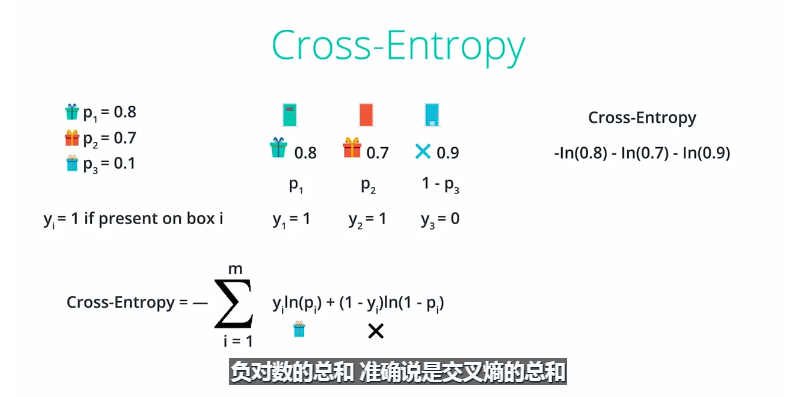
In this lesson and quiz, we will learn how to maximize a probability, using some math. Nothing more than high school math, so get ready for a trip down memory lane!

利用误差函数，用求和的方式来获取最大化概率的方式

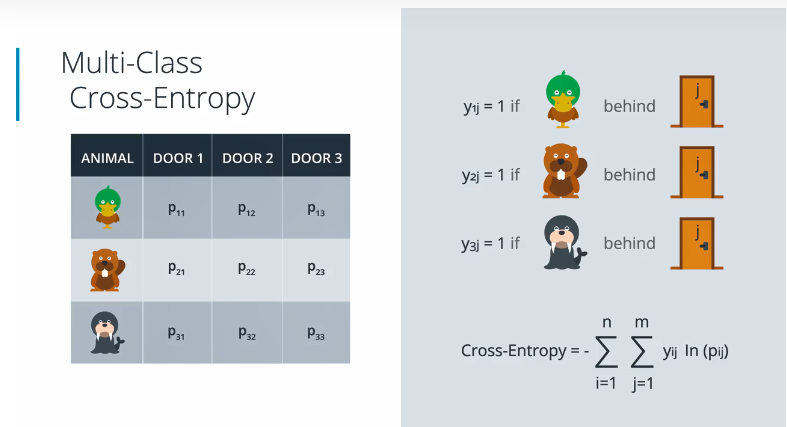
Cross-Entropy

用负对数来表示交叉熵，误差越大的模型，交叉熵越高；





多分类的交叉熵



Yij = 0/1

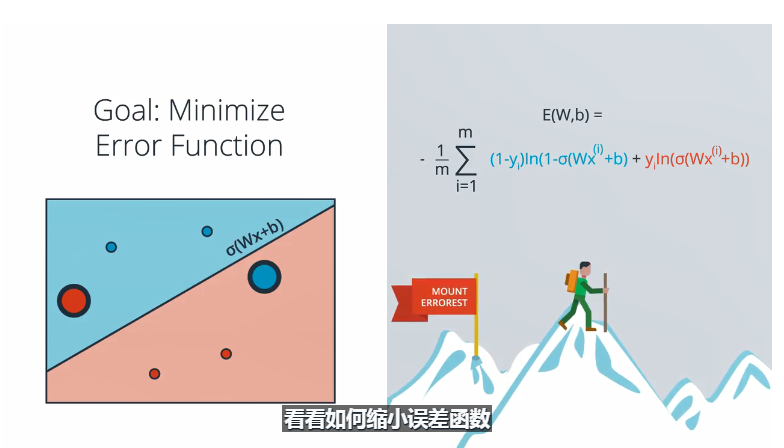
# Logistic Regression

Now, we're finally ready for one of the most popular and useful algorithms in Machine Learning, and the building block of all that constitutes Deep Learning. The **Logistic Regression** Algorithm. And it basically goes like this:

* Take your data
* Pick a random model
* Calculate the error
* Minimize the error, and obtain a better model
* Enjoy!

### Calculating the Error Function

Let's dive into the details. The next video will show you how to calculate an error function.



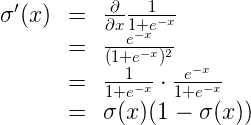
# Gradient Descent

# Gradient Calculation

In the last few videos, we learned that in order to minimize the error function, we need to take some derivatives. So let's get our hands dirty and actually compute the derivative of the error function. The first thing to notice is that the sigmoid function has a really nice derivative. Namely,

\sigma'(x) = \sigma(x) (1-\sigma(x))*σ*′(*x*)=*σ*(*x*)(1−*σ*(*x*))

The reason for this is the following, we can calculate it using the quotient formula:

[[](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)

And now, let's recall that if we have m*m* points labelled x^{(1)}, x^{(2)}, \ldots, x^{(m)},*x*(1),*x*(2),…,*x*(*m*), the error formula is:

E = -\frac{1}{m} \sum\_{i=1}^m \left( y\_i \ln(\hat{y\_i}) + (1-y\_i) \ln (1-\hat{y\_i}) \right)*E*=−*m*1​∑*i*=1*m*​(*yi*​ln(*yi*​^​)+(1−*yi*​)ln(1−*yi*​^​))

where the prediction is given by \hat{y\_i} = \sigma(Wx^{(i)} + b).*yi*​^​=*σ*(*Wx*(*i*)+*b*).

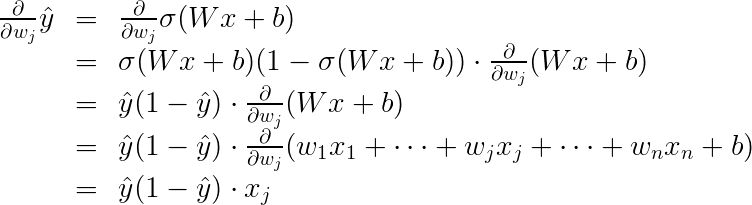
Our goal is to calculate the gradient of E,*E*, at a point x = (x\_1, \ldots, x\_n),*x*=(*x*1​,…,*xn*​), given by the partial derivatives

\nabla E =\left(\frac{\partial}{\partial w\_1}E, \cdots, \frac{\partial}{\partial w\_n}E, \frac{\partial}{\partial b}E \right)∇*E*=(∂*w*1​∂​*E*,⋯,∂*wn*​∂​*E*,∂*b*∂​*E*)

To simplify our calculations, we'll actually think of the error that each point produces, and calculate the derivative of this error. The total error, then, is the average of the errors at all the points. The error produced by each point is, simply,

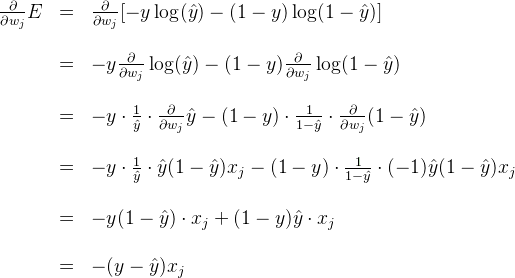
E = - y \ln(\hat{y}) - (1-y) \ln (1-\hat{y})*E*=−*y*ln(*y*^​)−(1−*y*)ln(1−*y*^​)

In order to calculate the derivative of this error with respect to the weights, we'll first calculate \frac{\partial}{\partial w\_j} \hat{y}.∂*wj*​∂​*y*^​.Recall that \hat{y} = \sigma(Wx+b),*y*^​=*σ*(*Wx*+*b*), so:

[[](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)

The last equality is because the only term in the sum which is not a constant with respect to w\_j*wj*​ is precisely w\_j x\_j,*wj*​*xj*​, which clearly has derivative x\_j.*xj*​.

Now, we can go ahead and calculate the derivative of the error E*E* at a point x,*x*, with respect to the weight w\_j.*wj*​.

[[](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)

A similar calculation will show us that

[[https://s3.cn-north-1.amazonaws.com.cn/u-img/ee078049-c5f4-4aee-b9c8-af8d15ced58c](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/5d171a72-bd11-4ea4-b138-4875a9c8d915/concepts/0d92455b-2fa0-4eb8-ae5d-07c7834b8a56)

This actually tells us something very important. For a point with coordinates (x\_1, \ldots, x\_n),(*x*1​,…,*xn*​), label y,*y*, and prediction \hat{y},*y*^​, the gradient of the error function at that point is \left(-(y - \hat{y})x\_1, \cdots, -(y - \hat{y})x\_n, -(y - \hat{y}) \right).(−(*y*−*y*^​)*x*1​,⋯,−(*y*−*y*^​)*xn*​,−(*y*−*y*^​)). In summary, the gradient is

\nabla E = -(y - \hat{y}) (x\_1, \ldots, x\_n, 1).∇*E*=−(*y*−*y*^​)(*x*1​,…,*xn*​,1).

If you think about it, this is fascinating. The gradient is actually a scalar times the coordinates of the point! And what is the scalar? Nothing less than a multiple of the difference between the label and the prediction. What significance does this have?

# Gradient Descent Step

Therefore, since the gradient descent step simply consists in subtracting a multiple of the gradient of the error function at every point, then this updates the weights in the following way:

w\_i' \leftarrow w\_i -\alpha [-(y - \hat{y}) x\_i],*wi*′​←*wi*​−*α*[−(*y*−*y*^​)*xi*​],

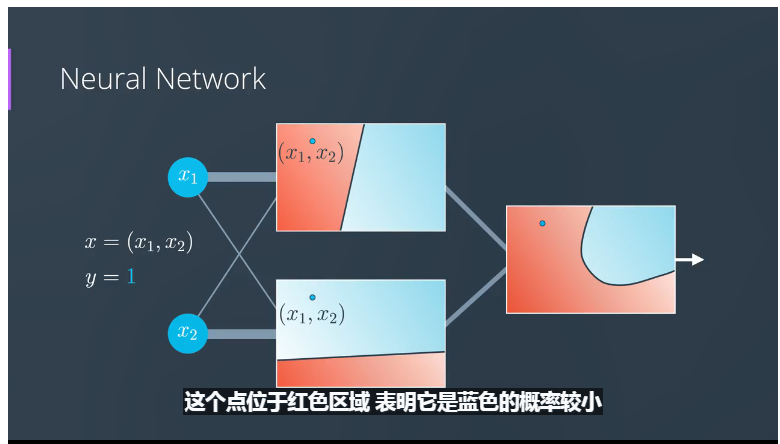
which is equivalent to

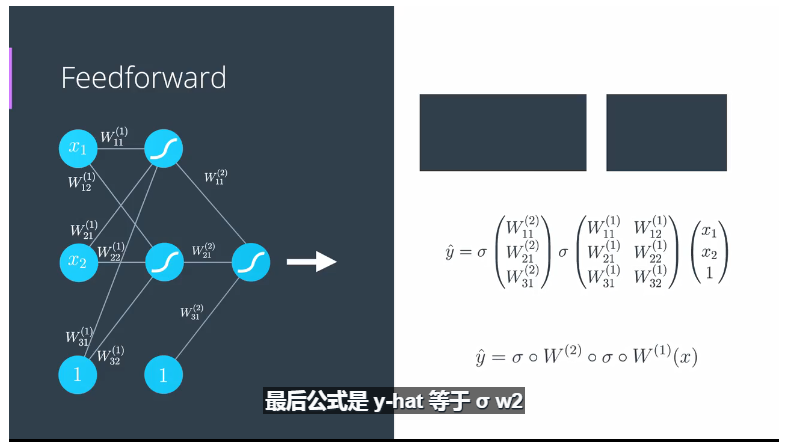
w\_i' \leftarrow w\_i + \alpha (y - \hat{y}) x\_i.*wi*′​←*wi*​+*α*(*y*−*y*^​)*xi*​.

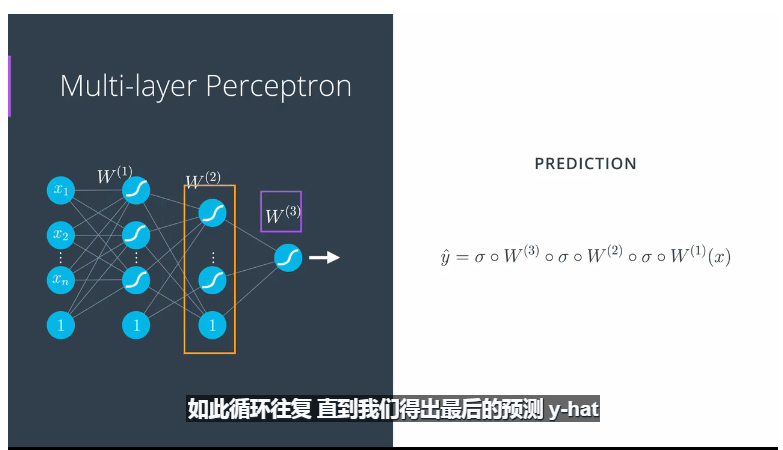
Similarly, it updates the bias in the following way:

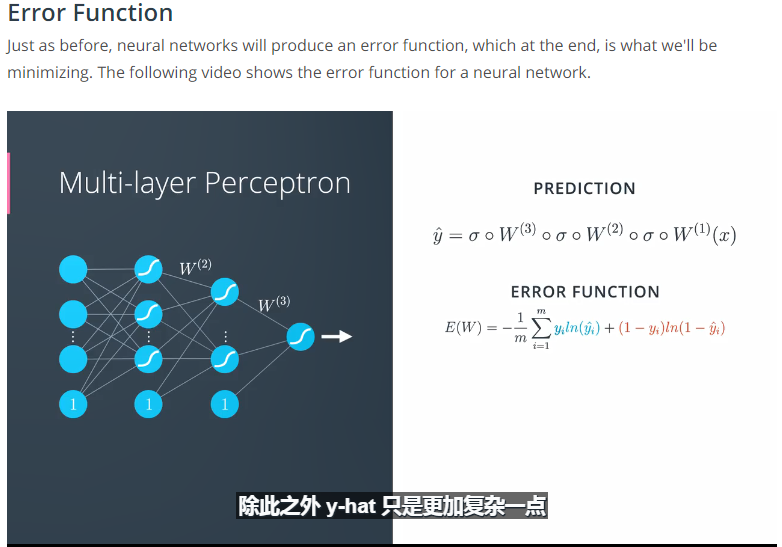
b' \leftarrow b + \alpha (y - \hat{y}),*b*′←*b*+*α*(*y*−*y*^​),

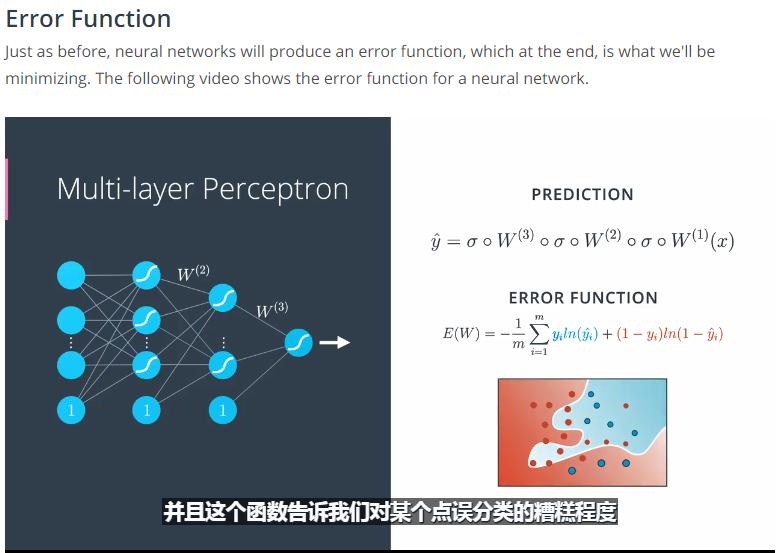
Note: Since we've taken the average of the errors, the term we are adding should be \frac{1}{m} \cdot \alpha*m*1​⋅*α* instead of \alpha,*α*,but as \alpha*α* is a constant, then in order to simplify calculations, we'll just take \frac{1}{m} \cdot \alpha*m*1​⋅*α* to be our learning rate, and abuse the notation by just calling it \alpha.*α*.

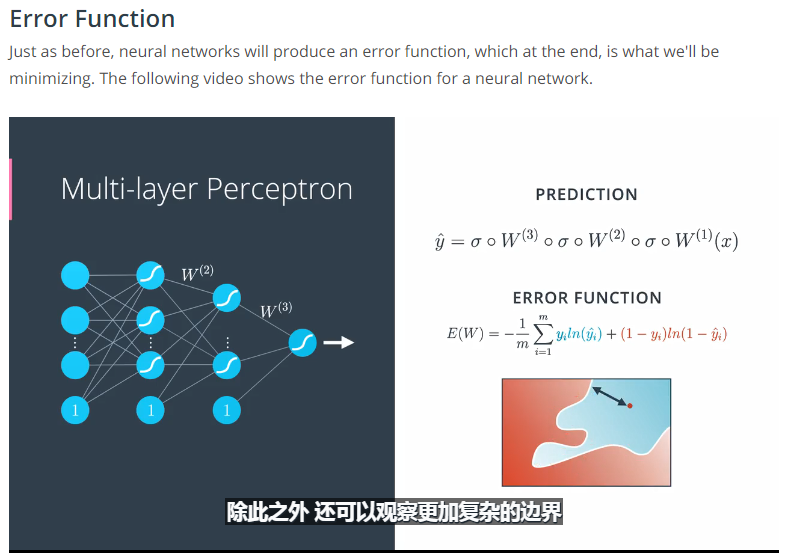


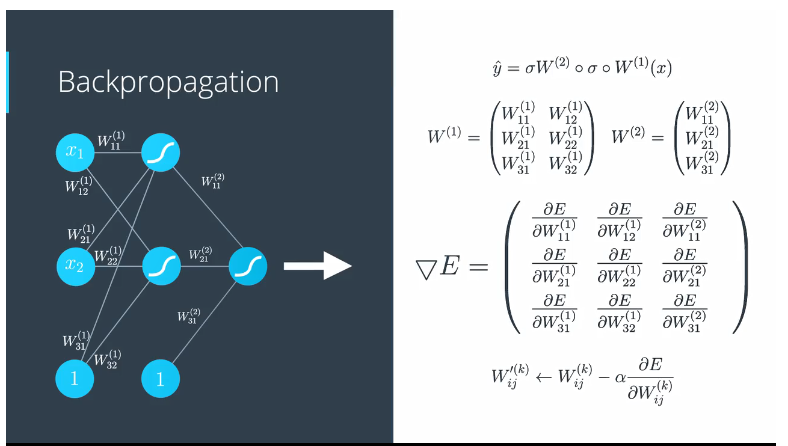


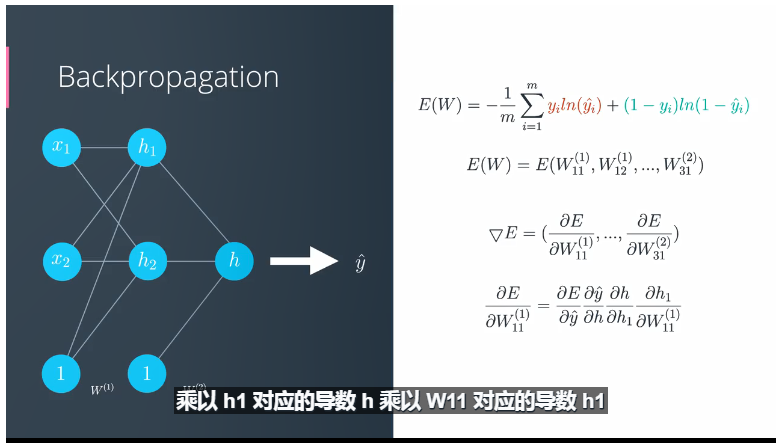












<https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.vt3ax2kg9>

<https://www.youtube.com/watch?v=59Hbtz7XgjM>