Normalization is scaling the data.

In logistics regression, the z is decision boundary, like z=wx+b like this or z might me circle equatin so if z is less than zero so sigmoid value will be lesser than 0.5 so it will predict y hat 0 and if z value is greater than 0 so its value is greater than 0.5 and will predict y hat 1

Lost function is for single training example, but cost function is for whole training set.

The cost function for logistic regression does not make bowl graph as made for linear regression. Because here fx is not a linear function it is non-linear function.

in course 2, will see automatic features selections.

Regularization is for avoiding overfitting, it is technique to reduce the size of feature by using value of w vector. We use this with lamda term. If lamda is zero so w will be large and it is overfit model and if lamda is very large, so w is so small that fx equal to b, it is horizontal straight line so choose lamda in between.

How to avoid overfitting:1 collect more data, 2 select features,3 regularization.

The answer of logistics regression is not 0 and 1 but it range from 0 and 1 then we apply condition that if its above 0.5 or equal so make it 1 otherwise 0.

In nurone layers, it takes input as 2d array, make it always like 1 by 2d array, even if you have 2 values so don't make 1d vector of 2 values.

And numpy and tensor or two different formats but they can be convertible.

- The `model.compile` statement defines a loss function and specifies a compile optimization.
- The `model.fit` statement runs gradient descent and fits the weights to the data.

Using matmul function after transposing, will be good practice.

Epochs in tensor flow code, is number of steps for gradient descent learn the model.

binaryCrossEntropy loss function is same as logistics regression function taught in course 1.

MeanSquareError is that one of linear regression.

Tensorflow use better algorithm than Gradient Descent

ReLU value from 0 to z for negative z is 0 and for any z input is same output z answer.

Linear activation is same as no activation function, same input give same output z=f(z)

Relu is faster than sigmoid, so mostly used in hidden layers. And for output layers its depends on project.

*LeakyReLU activation functions.

The linear function of linear function is itself a linear function. So, if all the layers are Linear activation so output layer will also be linear function of input layer directly. So that's why it is not useful to do computation with many layers of linear activation, we can just use directly input to output layers.

And that's why for output layers sigmoid we use ReLU hidden layers.

Why Non-Linear Activations?

The function shown is composed of linear pieces (piecewise linear). The slope is consistent during the linear portion and then changes abruptly at transition points. At transition points, a new linear function is added which, when added to the existing function, will produce the new slope. The new function is added at transition point but does not contribute to the output prior to that point. The non-linear activation function is responsible for disabling the input prior to and sometimes after the transition points. The following exercise provides a more tangible example.

The exercise will use the network below in a regression problem where you must model a piecewise linear target.

The sigmoid is best for on/off or binary situations. The ReLU provides a continuous linear relationship. Additionally, it has an 'off' range where the output is zero. The "off" feature makes the ReLU a Non-Linear activation. Why is this needed? Let's examine this below.

The goal of this exercise is to appreciate how the ReLU's non-linear behavior provides the needed ability to turn functions off until they are needed. Let's see how this worked in this example. The plots on the right contain the output of the units in the first layer.
Starting at the top, unit 0 is responsible for the first segment marked with a 1. Both the linear function z[illegible] and the function following the ReLU a[illegible] are shown. You can see that the ReLU cuts off the function after the interval [0,1]. This is important as it prevents Unit 0 from interfering with the following segment.
Unit 1 is responsible for the 2nd segment. Here the ReLU kept this unit quiet until after x is 1. Since the first unit is not contributing, the slope for unit 1, w[1]1[illegible]1[1], is just the slope of the target line. The bias must be adjusted to keep the output negative until x has reached 1. Note how the contribution of Unit 1 extends to the 3rd segment as well.
Unit 2 is responsible for the 3rd segment. The ReLU again zeroes the output until x reaches the right value.The slope of the unit, w[1]2[illegible]2[1], must be set so that the sum of unit 1 and 2 have the desired slope. The bias is again adjusted to keep the output negative until x has reached 2.

The "off" or disable feature of the ReLU activation enables models to stitch together linear segments to model complex non-linear functions.

Question: negative value of ReLU is zero so why this all, deep algorithm understanding for RelU I want

Softmax function for multiclass regression. And use SparsCatagoricalCrossEntroy function for loss.

Input argunment into loss functions CrossEntropy and SparsCataCrossEntropy (from_logits=true) is better numerically stable way. With output layer activation value to linear from instead of sigmoid or softmax then in the and separately use sigmoid or etc for problem.

Multiclass is different as what is multi-label classification.

In both softmax regression and neural networks with Softmax outputs, N outputs are generated, and one output is selected as the predicted category. In both cases a vector **z**$\hat{}$ is generated by a linear function which is applied to a softmax function. The softmax function converts **z**$\hat{}$ into a probability distribution as described below. After applying softmax, each output will be between 0 and 1 and the outputs will add to 1, so that they can be interpreted as probabilities. The larger inputs will correspond to larger output probabilities.

- SparseCategorialCrossentropy: expects the target to be an integer corresponding to the index. For example, if there are 10 potential target values, y would be between 0 and 9.
- CategoricalCrossEntropy: Expects the target value of an example to be one-hot encoded where the value at the target index is 1 while the other N-1 entries are zero. An example with 10 potential target values, where the target is 2 would be [0,0,1,0,0,0,0,0,0,0].

SOFT MAX LABS AGAIN WATCH

In machine learning neuro network, the cost function is found only at output layer of it is found for every neuron of every layer ???

In a neural network, the cost function, also known as the loss function, is typically calculated at the output layer. This cost function quantifies the difference between the predicted output of the network and the actual target output (i.e., the ground truth). The goal of training the neural network is to minimize this cost function, thereby improving the accuracy of the network's predictions.

However, during the training process, gradients are computed not only at the output layer but also at each neuron of every layer. This process is known as backpropagation. Backpropagation involves propagating the error gradients backward through the network, layer by layer, in order to adjust the parameters (weights and biases) of the network via gradient descent or its variants.

So, while the cost function is evaluated at the output layer to measure the overall performance of the network, the gradients needed for parameter updates are computed at every neuron of every layer during the backpropagation process. This allows the network to learn from the errors and improve its performance iteratively through training.

We find dJ/dw1 dJ/db1 dJ/dw2 dJ/db2 like means for every neuron's w and b changes how much effects cost function J(w,b).

High bias means how much output value is away from required output, and vice versa.

High variance means how much data is scattered like, if we input same thing into model, it gives every time different answer for that. It happens if Jcv error is high.

Large neuro networks lessen the high bias issue. And for high variance do add more data set.

Data augmentation for Adding More Data

Data Centric Approach, Model Centric Approach

Transfer learning. – supervised pretraining –fine tuning.

Error Analysis: It manually examine training sets that are mis classified.

Course 2 week 3, skewed datasets left for future.

Decision tree, purity term, splits that's how one sets has 5/5 cats and other has 0/5 means all dogs, so split like that or close to that.