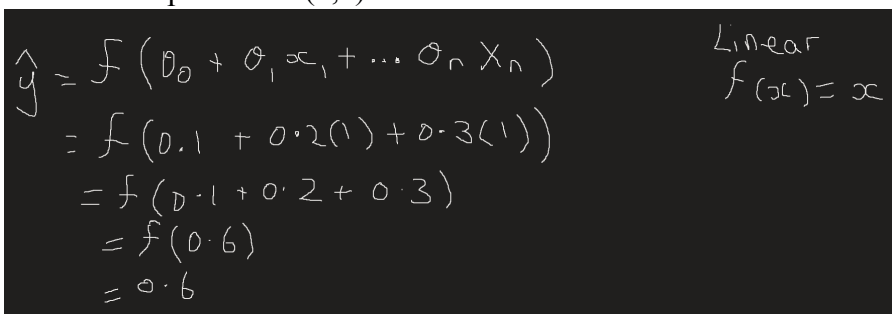
cs577 Assignment 2

Girish Rajani-Bathija
A20503736
Department of Computer Science
Illinois Institute of Technology
October 4, 2022

Artificial Neurons

1. Given a neuron with linear activation and weights (0.1, 0.2, 0.3) compute the output for the input vector (1,1).

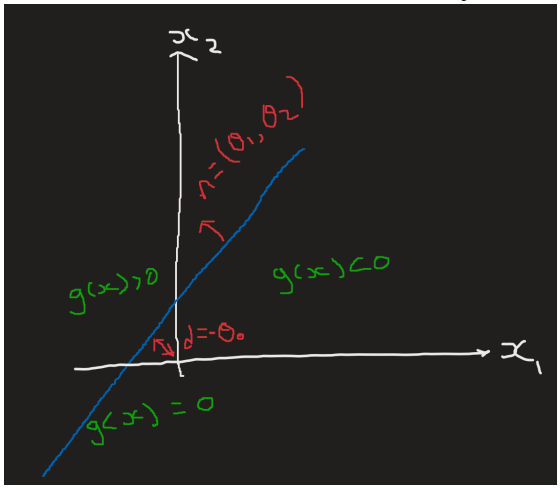


2. Given a linear discriminant function g(***x***) explain what should be its value on one side of the decision boundary, on the other side of the decision boundary, and on the decision boundary.

The decision boundary is defined by the function g(***x***). On one side of the decision boundary, it is positive so g(***x***) > 0, on the other side of the decision boundary, it is negative so g(***x***) < 0, and on the decision boundary itself, g(***x***) = 0

3. Given a linear discriminant function g(x_1, x_2) = $\theta_0 + \theta_1 x_1 + \theta_2 x_2$, explain the meaning of the coefficient $\theta_0\theta_1\theta_2$.

θ_1 and θ_2 are the coefficients of a normal vector that define the decision boundary and θ_0 defines the distance of the decision boundary from the origin. This can be seen below.



4. Let $g(x_1, x_2) = 1 + 2x_1 + 3x_2$. Find the normal to the decision boundary defined by $g(x_1, x_2)$ and the distance of the decision boundary from the origin (note that for the distance computation the normal has to be normalized).

The normal to the decision boundary is the normal vector θ_1 and θ_2 which is (2, 3) and the distance of the decision boundary from the origin is defined as θ_0 which is 1.

Note: we need to normalize the normal vector if it is not a unit vector, meaning if it's magnitude is not 1.

$|v| = \sqrt{(2^2 + 3^3)} = \sqrt{(4+9)} = \sqrt{(13)} = 3.6 \neq 1$

Therefore we need to normalize the normal.

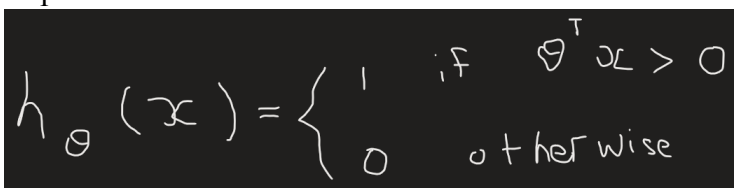
Normal vector (v) = <2,3>
Normalized normal vector (n) = n/|v| = $<2/\sqrt{13}, 3/\sqrt{13}>$

5. When writing a discriminant function as $g(x) = \theta^T x$ where x and θ are vectors, explain where the bias coefficient in θ is and what is the required change needed to the feature vector x to allow writing the discriminant in this way.

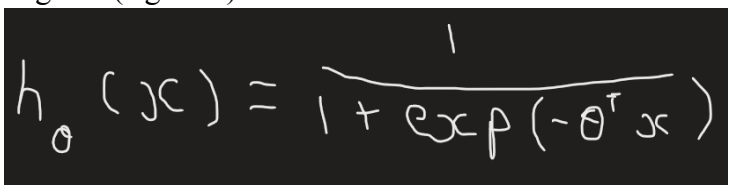
The bias coefficient in θ is θ_0 which is the distance from the origin and we hide the bias inside the coefficient so in the x vector you augment it so that it has 1 in the beginning, and in vector θ you concatenate θ_0 in the beginning. That is what is meant by hiding the bias inside the coefficient. As mentioned, the required change needed to the vectors is concatenating the θ_0 in the beginning of the θ vector and the 1 in the beginning of the x vector. This can be seen below:

6. Write the expressions for the step and logistic (sigmoid) activation functions. Explain the advantage of the sigmoid activation over step activation. What happens to the sigmoid when θ is small?

Step Activation:



Logistic (sigmoid) activation:

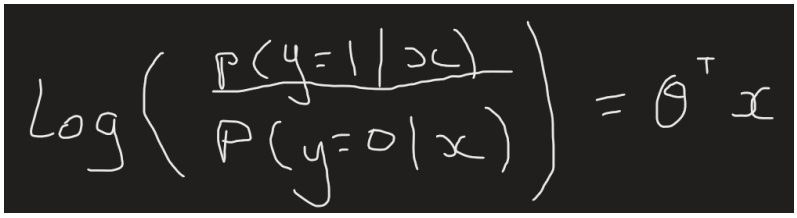


The advantage of using the sigmoid activation over step activation is that the step activation is too sharp in that the decision is either categorized as 0 or 1. It does not cater to uncertainty. Sigmoid is more gradual than sharp when compared to the step function. With sigmoid activation, instead of outputting only 0 or 1, it can also output a probability when there is uncertainty. For example, if the output is 0.8 for class 1, it could mean that it is not exactly in class 1 but there is an 80% chance that it is in class one. Another example, if the probability of a decision on the decision boundary is 0.5 then there is uncertainty as it has the same probability that it belongs to both class 1 and class 2. This results in a more accurate classification when compared to step activation, which categorizes the decision as either 0 or 1.

In sigmoid, when θ is small, then the sigmoid is considered to be 'soft'. This shows a lot of uncertainty as the values are not fixed and the model won't know exactly where the output belongs, but as training continues and becomes more certain then the θ increases and can make confident decisions whether it is 0 or 1.

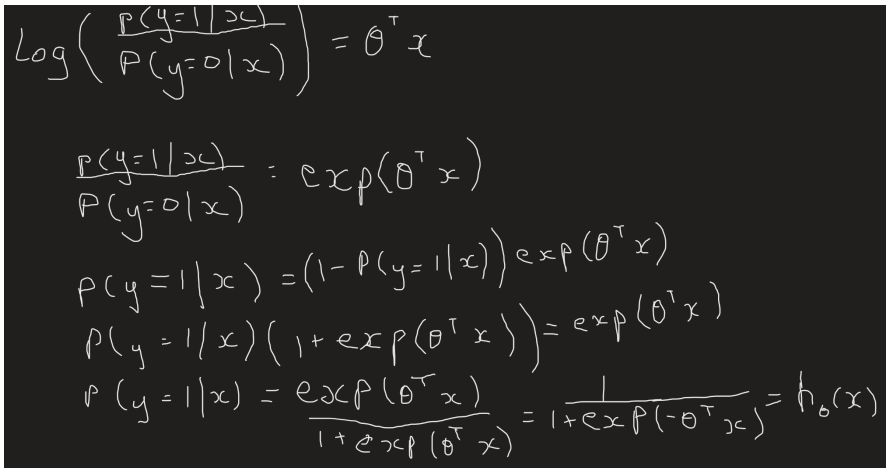
7. Explain how the sigmoid is obtained using a linear function to model the log-likelihood ratio.

The sigmoid function can be obtained by using the following linear function to model the log-likelihood ratio:



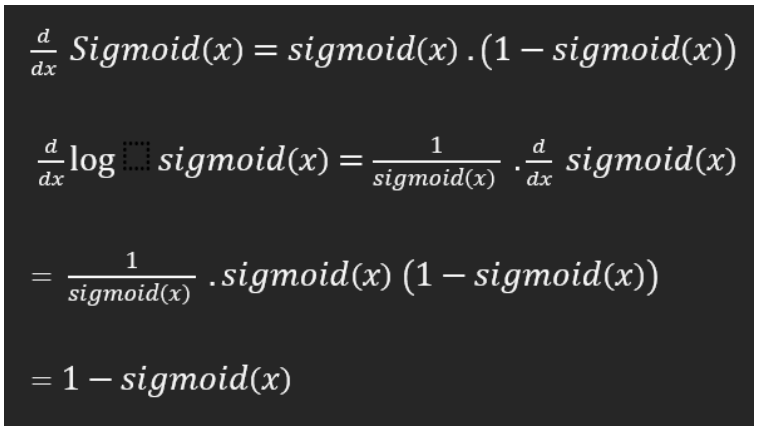
In this case, the log-likelihood ratio is modeled using the linear function, θ^Tx, which can then be used to get the sigmoid function.

The above equation is rearranged and the output of the log-likelihood function will be a probability that y = 1, given x, and this will be the same as the sigmoid activation function. This can be seen below in how the sigmoid is obtained.



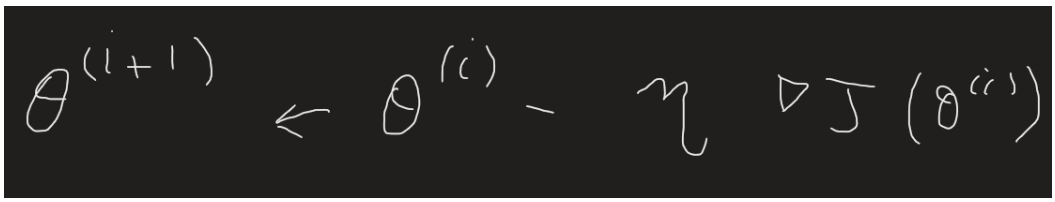
As shown above, the likelihood function and linear function are taken and multiplied by log on both sides, and then some rearranging and transposing are done. The final outcome is the sigmoid function. This is how the sigmoid is obtained using a linear function to model the log-likelihood ratio.

8. Write the derivative of a sigmoid and use it to compute the derivative of log-sigmoid.



9. Explain how to compute the direction used for updating parameters in gradient descent. How is the size of the update controlled?

The direction used for updating the parameters is computed by the following formula:



Where θ^(i) is the initial guess then you subtract the learning* the gradient of the initial guess.

Since the gradient points towards the direction of maximum change then we want to go in the opposite direction to reduce the loss and that is why we subtract the gradient from the initial guess to get the direction used for updating parameters in gradient descent. This formula is repeated until the gradient descent reaches the bottom and converges.

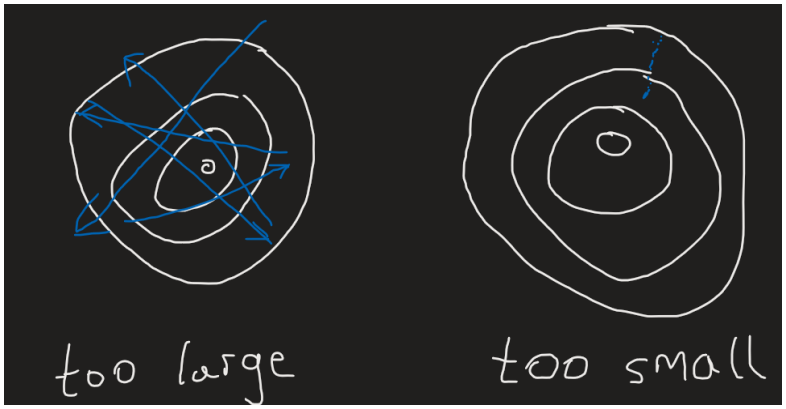
The size of the update is controlled by the learning rate. This will help to decide what the step size is for the gradient descent. If the learning rate is very small then the gradient descent will take very small steps and if the learning rate is very large then the gradient descent will take very large steps. This is explained further in question 11 below.

10. Explain the stop condition for gradient descent. Should the condition use the loss change or parameter change? Explain your answer.

Once the gradient descent knows the direction, it should go, and the step size, then it needs to know the stop condition. That is, when has the gradient descent reached the bottom and should stop. The stop condition for gradient descent can be looked at from both the loss change and parameter change. It is better if the stop condition uses the loss change instead because with the parameter change or change in theta, we do not know the value of the threshold parameter, and if we try and use some value, then we do not know how sensitive the theta is and this can impact the loss function in some way we do not want it to. With the loss change, we can easily say that since the loss is known, if the loss does not change much, the gradient descent can stop.

11. Explain what happens when using a learning rate that is too large or too small.

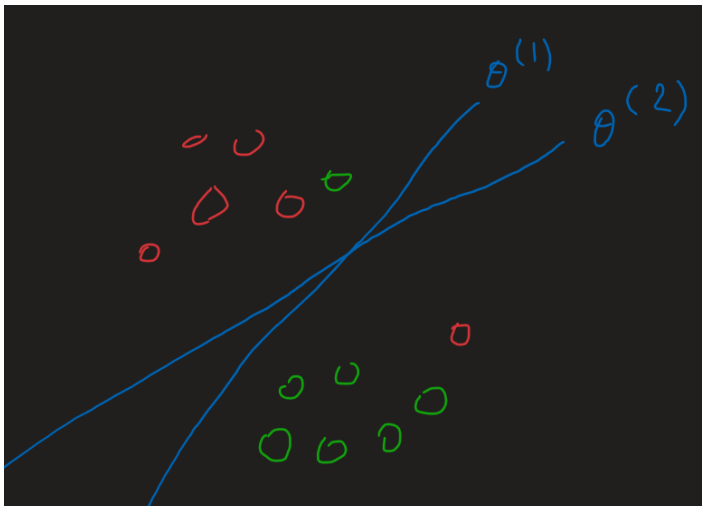
When the learning rate is too small, you will end up using a large number of iterations but still, the gradient descent does not get anywhere close to the bottom. This means that the training will happen very slowly as small updates are made after each iteration. However, if the learning rate is too large, then this can result in an unstable training process, as even though the optimal direction has been identified, the gradient descent keeps overshooting and will not converge. When the loss function is plotted, if the loss is fluctuating/zig-zagging, then the learning rate is too large. Both a too large and too small learning rate can be illustrated below.



12. Explain the empirical error loss is computed. What is the problem in using the empirical error loss in gradient descent?

The empirical error loss is calculated by counting the number of errors made with the goal of getting the error loss to 0. The number of errors made refers to how many times there were mistakes in classifying some data between some classes. For example, let us take 2 classes, class 0 and class 1. There will be 2 error sets. The first one will count how many examples of class 0 were classified in class 1 on error, and the second will count how many examples of class 1 were classified in class 0 on error. The empirical error loss is then the total of these two errors.

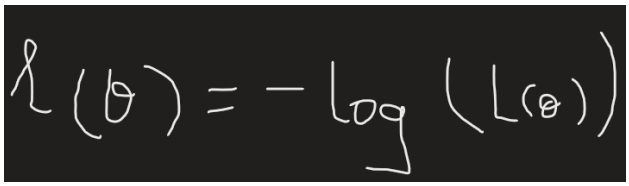
The problem in using the empirical error loss in gradient descent is that although it does a good job counting the number of errors made, it is a piece wise constant which means that the error cannot be reduced in gradient descent. Let us look at why this is which can be seen below.



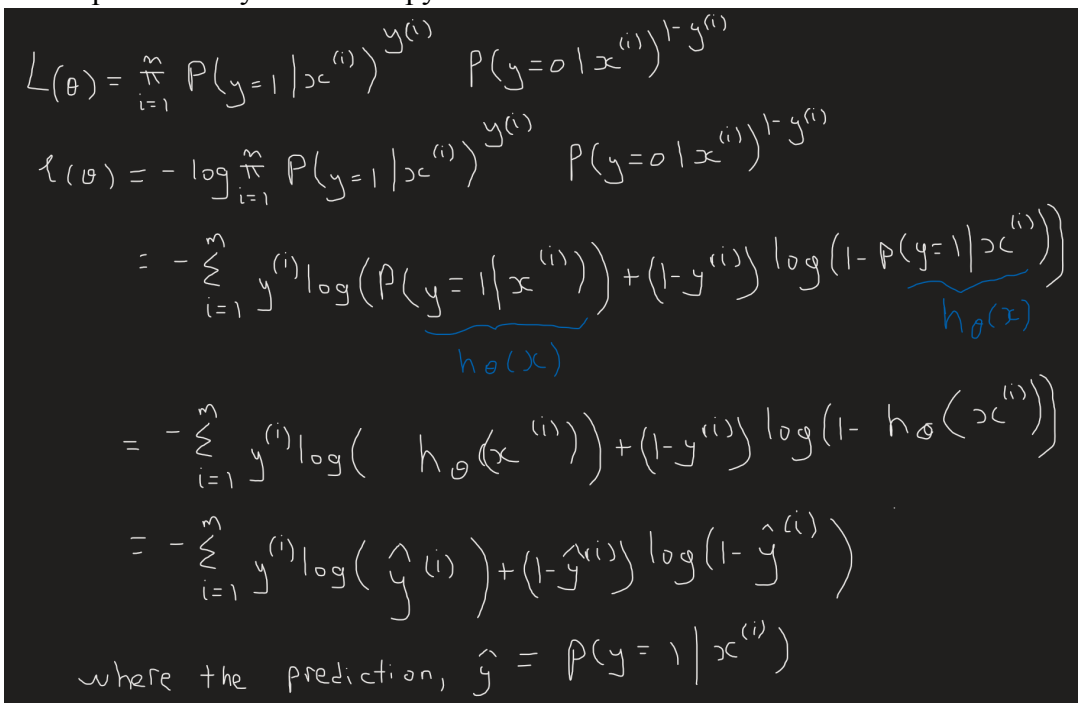
We can see that although we may try and change theta in the gradient descent to reduce the empirical error loss, it is simply not possible to reduce the number of errors. It still remains the same after applying theta2.

13. Write the log likelihood for a binary classifier and show how to develop it into binary cross-entropy.

Log likelihood for binary classifier:



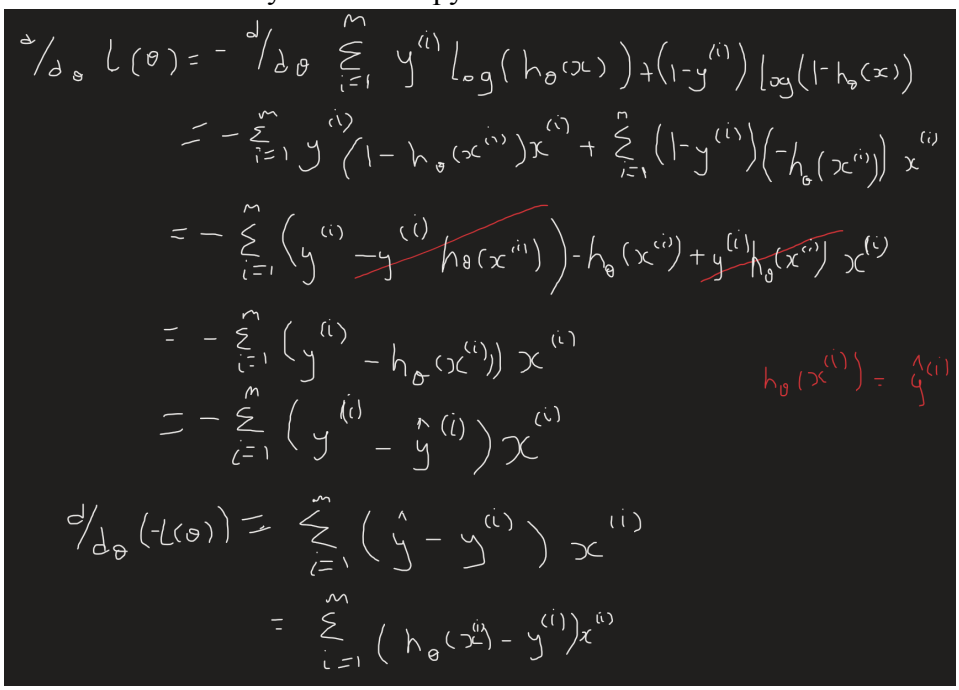
Develop into binary cross-entropy:



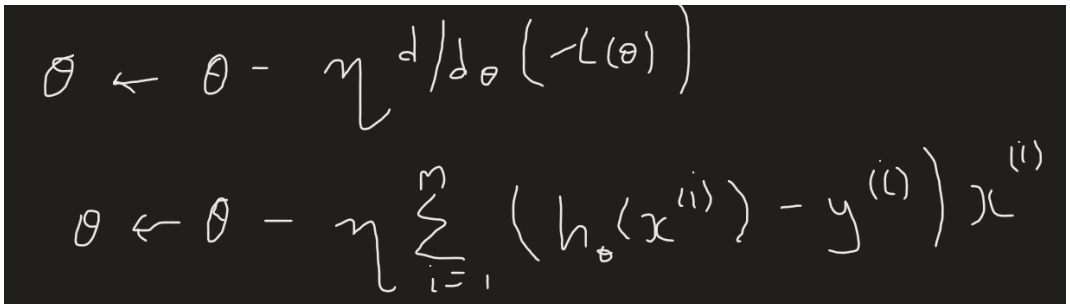
The above is the binary cross entropy because it is shown in the final version that there are 2 crosses for the prediction, so either the first term will be 0 and the second will be 1 or vice versa.

14. Write the gradient of the binary cross-entropy loss function with respect to the parameter vector when using a sigmoid activation function. Write the gradient descent update rule when using this loss function. Explain the meaning of the update equation you wrote.

Gradient of the binary cross-entropy loss function:



Gradient descent update rule:



The gradient descent update is the previous value - the learning rate * the derivative of the binary cross-entropy loss with respect to the parameter. The gradient descent update looks at the difference between the prediction (y (i) hat) and the label (y (i)), and whenever they agree (meaning both values are the same), there will be no update since the value will be 0 but when they disagree (meaning both values are different), the parameters are updated through the gradient descent update. Updating the parameters is done by adding the feature vector (x (i)) so the update is essentially the feature vector * the learning rate.

15. Explain the two strategies for multi-class classification (one against all others, one against each other). In which strategy is it easier to discriminate the data?

One against all others multi-class classification has to do with separating each class from all other classes by using individual neurons. Since a single neuron can only distinguish between two classes, with this strategy, a neuron is separating one class from all other classes combined. For example, if there are 3 classes, C1, C2, C3, the first neuron will distinguish C1 from all other 2 classes, the 2nd neuron will distinguish C2 from all other classes and the third neuron will do the same with C3. Each neuron will output a probability that it belongs to a class and then the one that has the highest probability is the class that it belongs to.

One against each other multi-class classification has to do with separating a class from each other. Instead of a single neuron separating a single class from every other classes, this time, a single neuron will separate a class from another class. For example, if there are three classes, C1, C2, C3, the first neuron will distinguish C1 from C2, the second neuron will distinguish C1 from C3 and the final neuron will separate C3 from C2.

One against each other is the strategy that is easier to discriminate the data.

16. Explain the template matching interpretation of linear discriminants. Using this interpretation what is the meaning of using multiple linear discriminant functions.

In linear discriminants, the template matching interpretation instead of separation, has to do with large similarity to K templates whereby the features such as the weights of a neuron are stacked on top of each other in a vector as templates. When using multiple linear discriminant functions, instead of separating the classes, it will compare the features to the templates and look at the similarity. If a sample is similar to a particular class in that it has high similarity to the template, then it will be classified in that class.

17. Explain the need for using softmax in the output layer of a multi-class classifier.

Softmax is used in the output layer of a multi-class classifier because if we use a sigmoid activation, then although the probabilities of the individual neurons will be between 0 and 1, the sum of the probabilities will be greater than which is not what we want. Instead, we want the sum of the probabilities of the output to be equal to 1. That is where the softmax activation function comes in. By making all the decimal probabilities add up to 1 when using softmax, this allows the training data to converge faster than when using sigmoid activation.

18. Write the derivative of softmax and use it to compute the derivative of log-softmax

$$\frac{\partial}{\partial \theta_i} \text{softmax}(\theta_j^T x) = \text{softmax}(\theta_j^T x)\left(\delta_{ij} - \text{softmax}(\theta_i^T x)\right) x$$

$$\frac{\partial}{\partial \theta_i} \log\left(\text{softmax}(\theta_j^T x)\right) = \frac{1}{\text{softmax}(\theta_j^T x)} \cdot \frac{\partial}{\partial \theta_i} \text{softmax}(\theta_j^T x)$$

$$= \frac{1}{\text{softmax}(\theta_j^T x)} \cdot \text{softmax}(\theta_j^T x)\left(\delta_{ij} - \text{softmax}(\theta_i^T x)\right) x$$

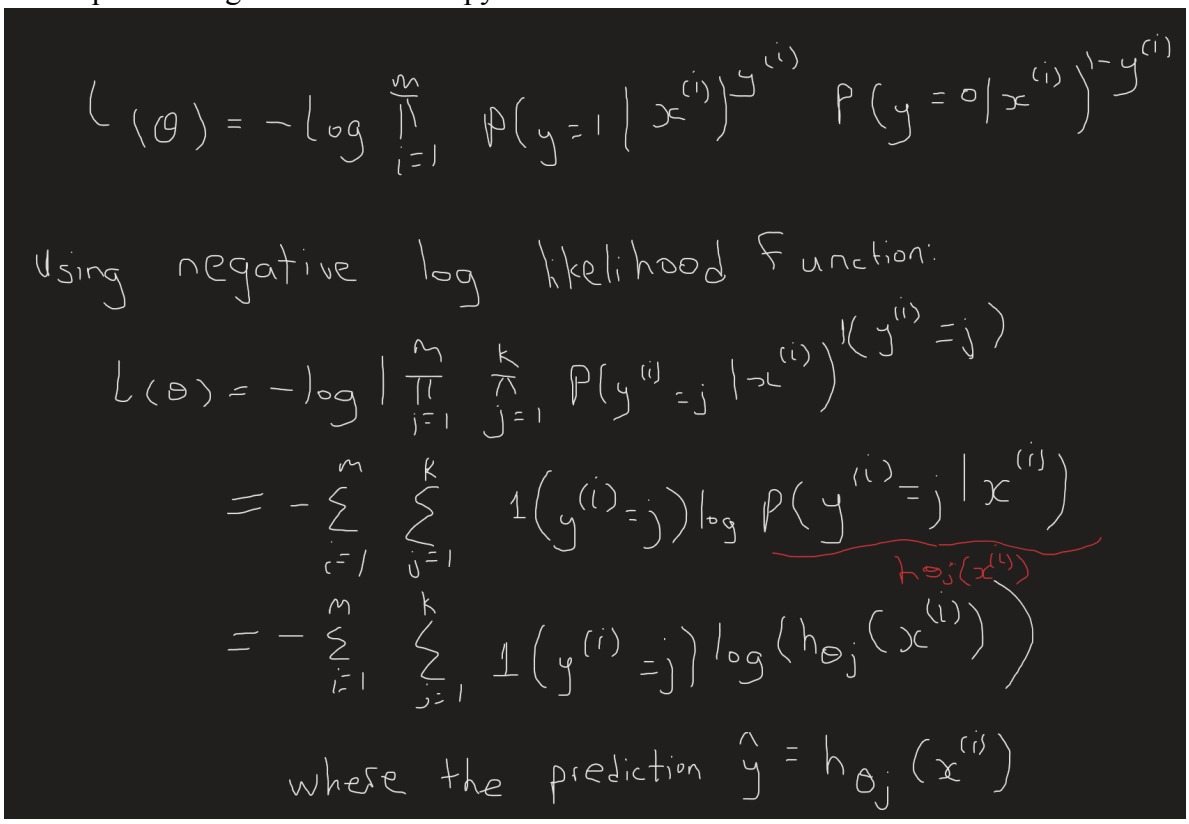
$$= \left(\delta_{ij} - \text{softmax}(\theta_i^T x)\right) x$$

19. Write the log likelihood for a multiclass classifier and show how to develop it into categorical cross-entropy.

Log likelihood for multiclass classifier:

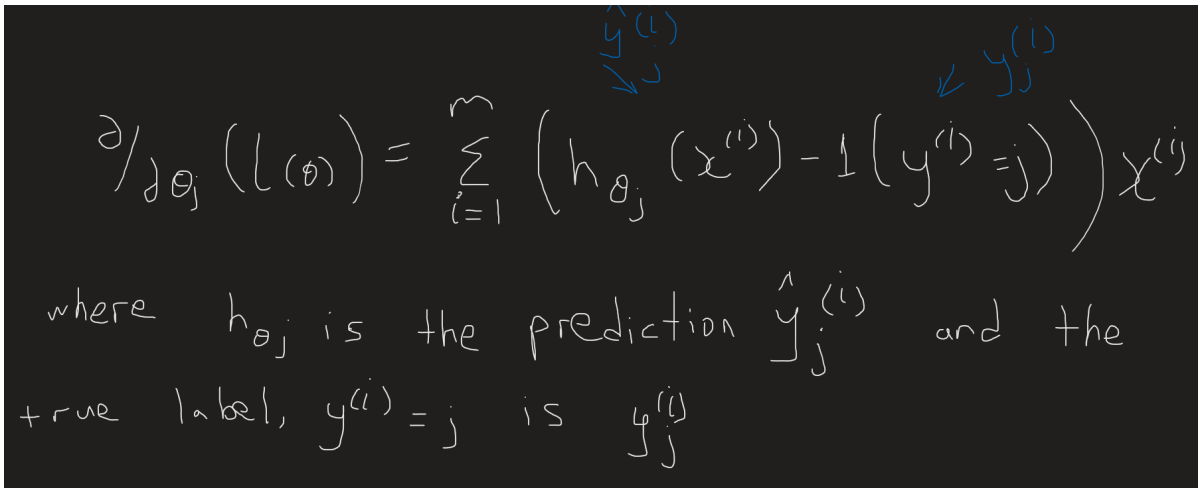
$$L(\theta) = -\log \prod_{i=1}^{m} P(y=1 \mid x^{(i)})^{y^{(i)}} P(y=0 \mid x^{(i)})^{1-y^{(i)}}$$

Develop into categorical cross-entropy:

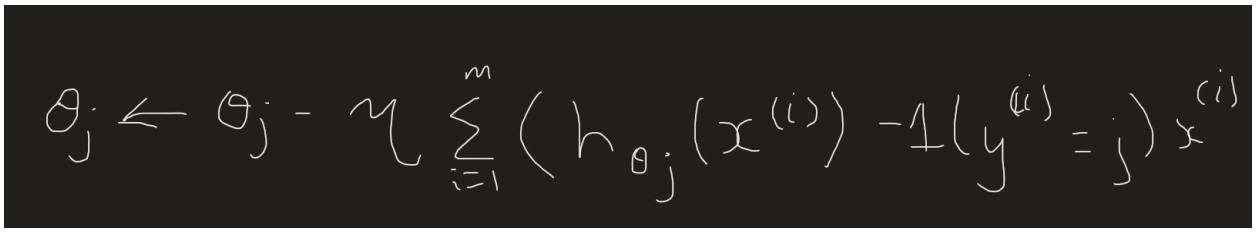


20. Write the gradient of the categorical cross-entropy loss function with respect to the parameter vector when using a softmax activation function. Write the gradient descent update rule when using the loss function. Explain the meaning of the update equation you wrote.

Gradient of the categorical cross-entropy loss function:



Gradient descent update rule:



The gradient descent update rule is the parameters of the j th neuron - the learning rate * by the sum of all the examples that were incorrectly classified * the feature vector. This will give the update to the parameters. What this does is that after training, the output of each neuron will be a probability then the neuron that has the highest probability will be the chosen class that is classified.

Neural Networks

1. Given a two layer network where the number of hidden units is larger than the number of inputs, explain the dimensionality increase interpretation of the hidden layer. Explain what is the possible benefit of such dimensionality increase.

When the number of layers is more than the number of inputs, dimensionality increase occurs where the dimensions of the input vector is increased to match the hidden layer unit which means that the input layer is moved to a higher dimension space. This is common in non linear models where the model is much more complex. The possible benefit of dimensionality increase is when we are working with non linear classification problems whereby using dimensionality increase can make solving this type of problem possible.

2. Given a two layer network where the number of hidden units is smaller than the number of inputs, explain the dimensionality decrease interpretation of the hidden layer. Explain what is the possible benefit of such dimensionality decrease.

When the number of hidden layers is less than the number of inputs, dimensionality reduction occurs. This means that the feature vectors in the dataset are reduced which results in fewer parameters focusing only on the most relevant variables. Dimensionality reduction can be a benefit depending on how much data you have the problem that is being solved. The possible benefit of dimensionality decrease is to simplify the data which is commonly done in linear models where by having a simpler model, generalization is increased.

3. Given the update equations for the output layer in a 2 layer network, explain why we cannot use directly the same equations for the hidden layer. Assume that both layers have the same activation function.

Firstly, let us look at the update equation for the output layer:

$$v_j \leftarrow v_j - \eta \sum_{i=1}^{m} \left(\hat{y}_j^{(i)} - 1(y^{(i)} = j) \right) z^{(i)}$$

Update equation for hidden layer:

$$w_j \leftarrow w_j - \eta \sum_{i=1}^{m} \left(\hat{z}_j^{(i)} - 1(z^{(i)} = j) \right) x^{(i)}$$

It can be seen that both are similar but slightly different. It is not possible to use the same equation of the output layer for the hidden layer because the input for both layers are different. The input for the hidden layer is x^(i) which is the input feature vector then the input for the output layer is z^(i) which is the output of the hidden layer.

Additionally, the parameters of the hidden layer weights (wj) and output layer weights (vj) are guessed and and used to compute the z(i) output of the hidden layer then that is used as input in the output layer. If the update equation is the same for both the hidden and output layer, then it will not be possible calculate z(i) and the parameters cannot be updated accurately.

4. Apply the chain rule to compute the gradients with respect to both hidden layer and output layer parameters of a 2-layer regression network with a single output where the hidden layers use sigmoid activation and the output layer uses linear activation.

To compute the gradients, we calculate the derivatives of the loss (E) with respect to the parameters of the output layer (v) and also computing the derivatives of the loss (E) with respect to each of the weights of the hidden layers (wj). This can be done by applying the chain rule:

$$\frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v}$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial w_j}$$

$$\frac{\partial E}{\partial v} = \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) z^{(i)}$$

$$\therefore v \leftarrow v - \eta \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) z^{(i)}$$

We can see that the update gradient equation for the output layer is shown by v and is computed from the derivative

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) v_j \, z_j^{(i)} \left(1 - z_j^{(i)} \right) x^{(i)}$$

$$\therefore w_j \leftarrow w_j - \eta \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) v_j \, z_j^{(i)} \left(1 - z_j^{(i)} \right) x^{(i)}$$

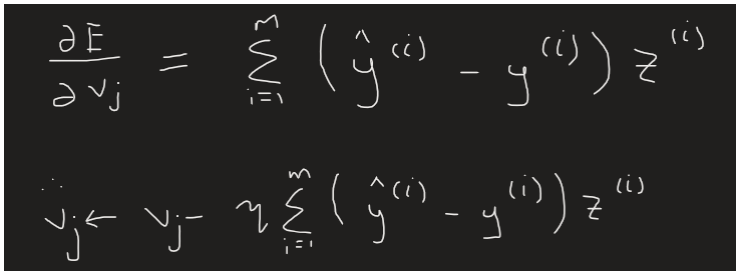
We can see that the update gradient equation for the hidden layer is shown by wj and is computed from the derivative

5. Apply the chain rule to compute the gradients with respect to both hidden layer and output layer parameters of a 2-layer regression network with multiple outputs where the hidden layers use sigmoid activation and the output layer uses linear activation.

To compute the gradients, we calculate the derivatives of the loss (E) with respect to the parameters of the output layers (vj) and also computing the derivatives of the loss (E) with respect to each of the weights of the hidden layers (wj). This can be done by applying the chain rule:

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial v_j}$$

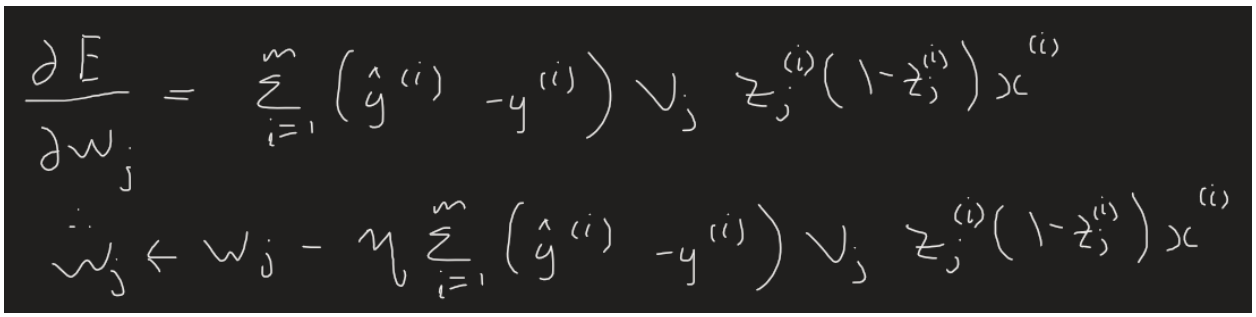
$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial w_j}$$



$$\frac{\partial E}{\partial v_j} = \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right) z^{(i)}$$

$$v_j \leftarrow v_j - \eta \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right) z^{(i)}$$

We can see that the update gradient equation for the output layers is shown by vj and is computed from the derivative

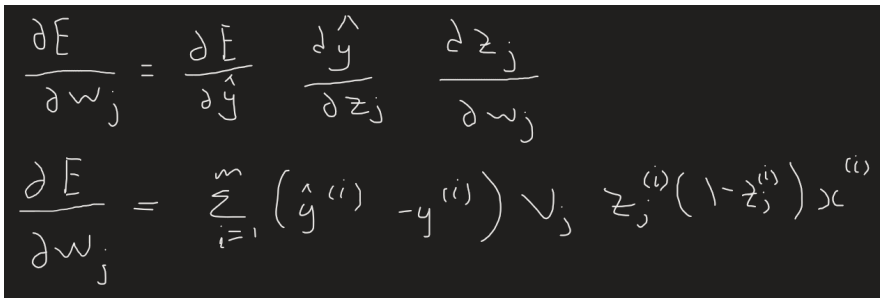


$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right) v_j \, z_j^{(i)} \left(1 - z_j^{(i)}\right) x^{(i)}$$

$$w_j \leftarrow w_j - \eta \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right) v_j \, z_j^{(i)} \left(1 - z_j^{(i)}\right) x^{(i)}$$

We can see that the update gradient equation for the hidden layers is shown by wj and is computed from the derivative

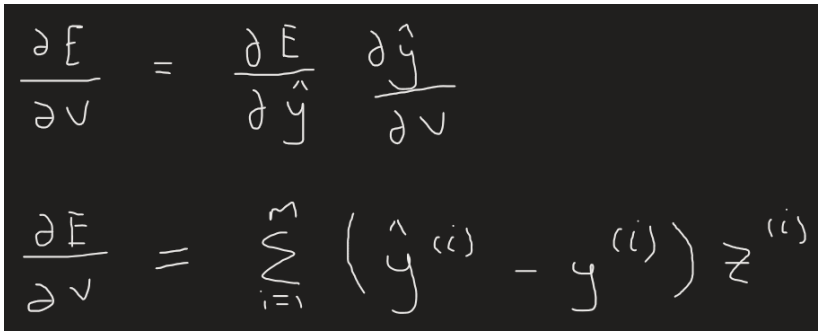
6. Apply the chain rule to compute the gradients with respect to both hidden layer and output layer parameters of a 2-layer binary classification network where the hidden layers use sigmoid activation and the output layer uses sigmoid activation.



$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial w_j}$$

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right) v_j \, z_j^{(i)} \left(1 - z_j^{(i)}\right) x^{(i)}$$

We can see the gradient is computed from the derivative with respect to the hidden layers (wj)

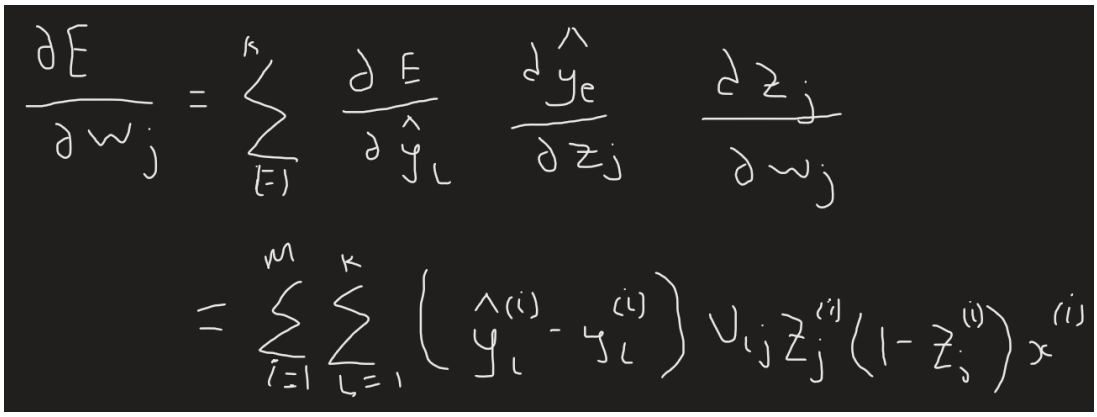


$$\frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v}$$

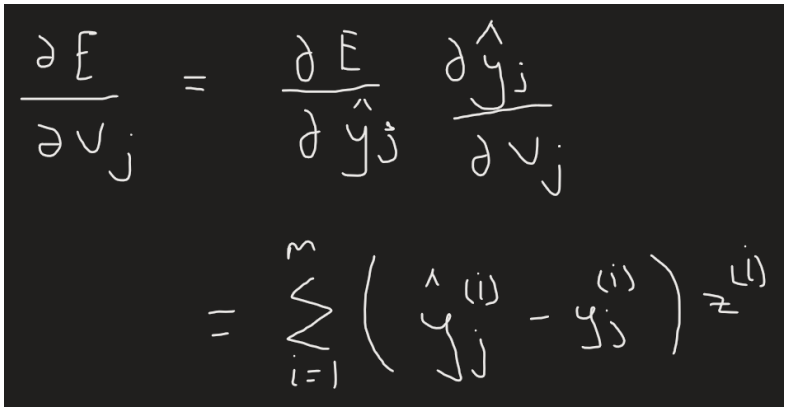
$$\frac{\partial E}{\partial v} = \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right) z^{(i)}$$

We can see the gradient is computed from the derivative with respect to the output layer (v)

7. Apply the chain rule to compute the gradients with respect to both hidden layer and output layer parameters of a 2-layer multiclass classification network where the hidden layers use sigmoid activation and the output layer uses softmax activation.



We can see the gradient is computed from the derivative with respect to the hidden layers (wj)



We can see the gradient is computed from the derivative with respect to the output layers (vj)

8. Describe how weights in the network should be initialized and explain why they are initialized in this way.

The weights in the network are initialized randomly using guesses at the start. Through forward pass, those guesses are pushed through the network to calculate the output for the hidden layer (z (i))and output layer (y (i) hat). Now that we know what the output predictions are, we can use those predictions to calculate the gradient update rule to update the weights (v and w) through each iteration.

It is important to initially guess the weights so that the output predictions can be calculated, and once those predictions are obtained, they can be used to update the weights in the network through each iteration. This is important because if we do not guess the weights, then we do not know these values and will not be able to update them. If we just initialize the weights using 0, then the neurons will just learn the same features of the inputs and will decrease the accuracy.

Computation Graphs

1. Explain the advantage of using computation graphs. Describe what is computed during the forward pass and backward pass of the networking during backpropagation. Explain what each node in the graph needs to be able to compute and what information it needs to store (and why).

The advantage of using computation graphs is that it is used to represent mathematical expressions in deep learning, and it is used to calculate the forward pass and backward pass. These can then be used to calculate the derivatives to find the gradient automatically. This is helpful because, in a network of multiple layers, it can be tedious to compute the derivatives manually. However, through computation graphs, the derivative of all the layers can automatically be computed.

During the forward pass, the values of the output (**or the function value**) at each neuron are computed from the input data. It pushes the input through the network, calculating the output from neuron to neuron until it reaches the output layer, which calculates the loss function. During the backward pass, the derivative of the output with respect to the input is calculated and this is passed back through the network to update the weights so that the loss can be minimized. The computation this time starts from the last layer and goes back until it reaches the first layer.

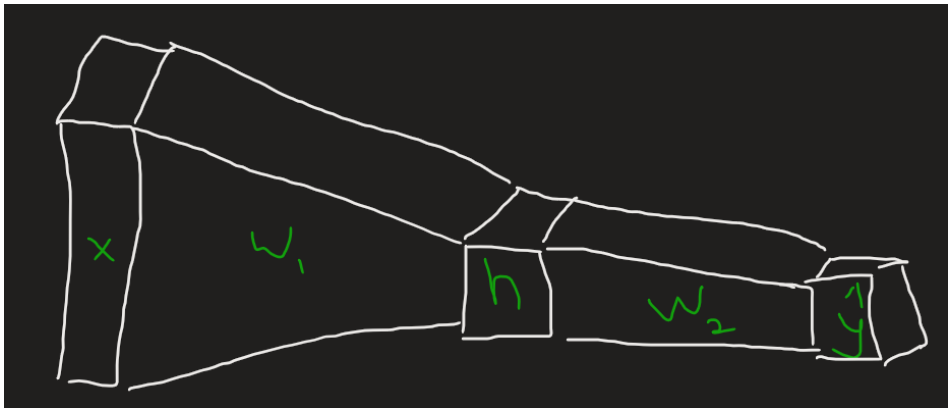
Each node during the forward pass needs to know how to take the input numbers x and y and compute the function value, which will output z. For backward pass, the node needs the derivative of the output to compute the two derivatives of the two inputs and then output those input values. The nodes will compute the local gradient and output the product of the incoming and local gradient.

Each node needs to store the x and y values (the input values) so that for the backward pass, it can compute the gradient of those values to push back through the network and for the input pass those values and multiplied to give the output. Therefore it is important to store those 2 values as they are needed for computation. For example, when doing backward pass, to compute one of the outputs dz/dx, we need to know the value of y and then when computing the other output dz/dy, we need to know the value of x.

2. Function composition:

   Y_hat = f2(W2, f1(W1,x))

3.
4. The computation graph will be 3-dimensional since we are assuming that the feature vectors are 3-dimensional:



The inputs of the computational graph are the input training feature vectors(x), the input training labels (y), and the weights of the nodes in the hidden layer (W1, and W2). The derivatives needed:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial a} \frac{\partial a}{\partial w_1}$$

Where dL is the loss gradient, dy_hat is the output prediction, da is the result after multiplying w1 and x inputs, and dh is the result after passing 'a' through the sigmoid activation function.

5. Describe the back flow patterns for the following nodes: addition of a constant, addition of two inputs, multiplication by a constant, multiplication of two inputs, max of two inputs, min of two inputs.

For the addition of a constant operation, the input x is added to a constant c within the node to output x+c. The derivative of the output with respect to the input (d/dx (x+c)) gives us a 1. This means that during backward pass, the incoming gradient is multiplied by 1 and this just means that the gradient is flowing through the network without a change (it will not get small or bigger). This is the expected result for all addition nodes, where the gradient will just flow through the network without a change.

For the addition of two inputs (x and y), the addition node will compute the sum of the inputs and output x+y and during backward pass, the derivative with respect to x is 1 and the derivative with respect to y is also 1. The incoming gradient will be distributed to the x and y but it will not change the value. The same value will just be passed to those inputs because, as explained previously, the addition nodes will not change the gradient.

For the multiplication by a constant, the input x is multiplied by the input constant c to get c*x. During backward pass, the gradient is only pushed in the direction of x since c is a gradient so there won't be any change to c. When the gradient is passed to x, it is multiplied by the constant c as when we find d/dx (c*x), we get c. This means that the gradient will keep multiplying by the constant c as it is going through the network which will keep increasing the gradient which results in the exploding gradient problem.

For the multiplication of two inputs, the derivative with respect to x is calculated which is d/dx (x*y) = y and the derivative with respect to y is calculated which is d/dy (x*y) = x. This means that during backward pass, when passing the gradient through the network, the gradient is multiplied by x as it flows through the y input (because the derivative with respect to y was x as mentioned earlier) and the gradient is multiplied by y as it flows through the x input.

For the max of two inputs, the max node finds the the greater value between x and y by saying if x >= y then the output is x, otherwise the output is y. After this, the derivative of x, d/dx max (x,y) = 1 if x >= y, otherwise 0 and the derivative of y, d/dy max (x,y) = 0 if x >= y, otherwise 1. If the derivative of x is 1 then the gradient will flow through x without change and it will be 0 towards the y input (no gradient will flow towards it). Similarly, if y > x where the derivative of y is 1 then the gradient will flow through the y input without change and nothing will flow through x.

For the min of two inputs, the min node finds the lower value between x and y by saying if x <= y then the output is x, otherwise the output is y. After this, the derivative of x, d/dx min (x,y) = 1 if x <= y, otherwise 0 and the derivative of of y, d/dy min (x,y) = 0 if x <=y. Otherwise 1. If the derivative of x is 1, then x is lower than y so the gradient will flow through x without change and it will be 0 towards the y input (no gradient will flow towards it). Similarly, if y < x where the derivative of y is 1 then the gradient will flow through the y input without change and nothing will flow through x.

6. Explain what derivative you expect when the input to a node is a vector and the output is a scalar. Write all the components of the derivative.

When the input to a node is a vector (let us say w) and the output is a scalar (let us say y_hat) then the derivative is dy_hat / dw. Since this is the derivative of a scalar with respect to a vector, the output will be a gradient vector whose rank is 1.

dy_hat/dw → gradient (vector) - Rank 1

When looking at the derivatives, the rank of the output must be the same as the rank of the dependent variable (y_hat) + the rank of the input variable. In the above example, the scalar y_hat has a rank of 0 and the vector w has a rank of 1 so 0 + 1 is 1, therefore, the rank of this derivative is a rank 1 tensor.

7. Explain what derivative you expect when the input to a node is a vector and the output is a vector. Write all the components of the derivative.

When the input to a node is a vector (let us say w) and the output is a vector (let us say y_hat) then the derivative is dy_hat / dw. Since this is the derivative of a vector with respect to a vector, the output will be a matrix whose rank is 2.

dy_hat/dw → Rank 2 tensor (matrix)

When looking at the derivatives, the rank of the output must be the same as the rank of the dependent variable (y_hat) + the rank of the input variable. In the above example, the vector y_hat has a rank of 1 and the vector w has a rank of 1, so 1 + 1 is 2, therefore, the rank of this derivative is a rank 2 tensor.

8. Explain what derivative you expect when the input to a node is a matrix and the output is a scalar. Write all the components of the derivative.

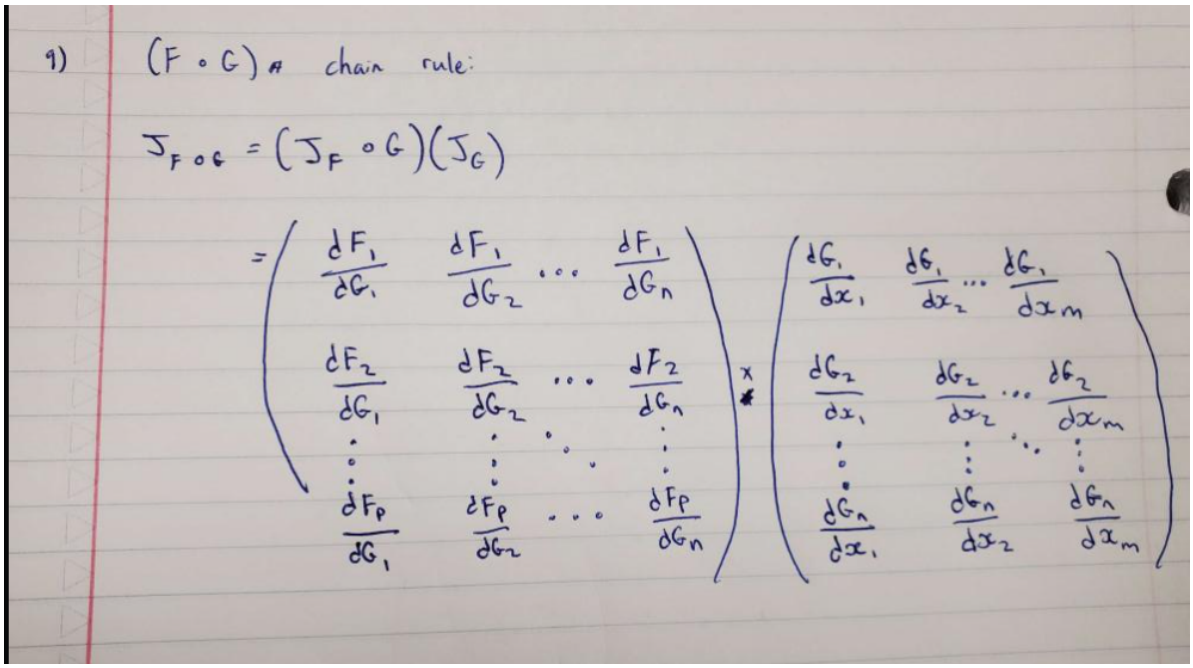
When the input to a node is a matrix (let us say x) and the output is a scalar (let us say y_hat) then the derivative is dy_hat / dx. Since this is the derivative of a matrix with respect to a scalar, the output will be a matrix whose rank is 2.

dy_hat/dx → Rank 2 tensor (2D matrix)

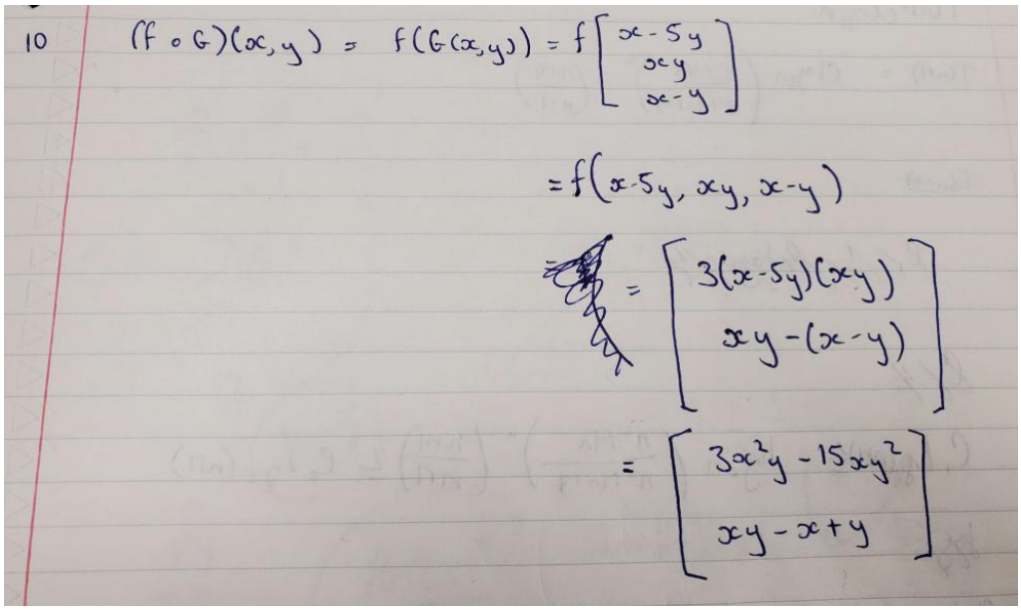
When looking at the derivatives, the rank of the output must be the same as the rank of the dependent variable (y_hat) + the rank of the input variable. In the above example, the scalar y_hat has a rank of 0 and the matrix x has a rank of 2, so 0 + 2 is 2, therefore, the rank of this derivative is a rank 2 tensor.

9. Write the chain rule for the composition of vector valued vector functions F and G. Pay attention to the order of the Jacobian matrices you write.

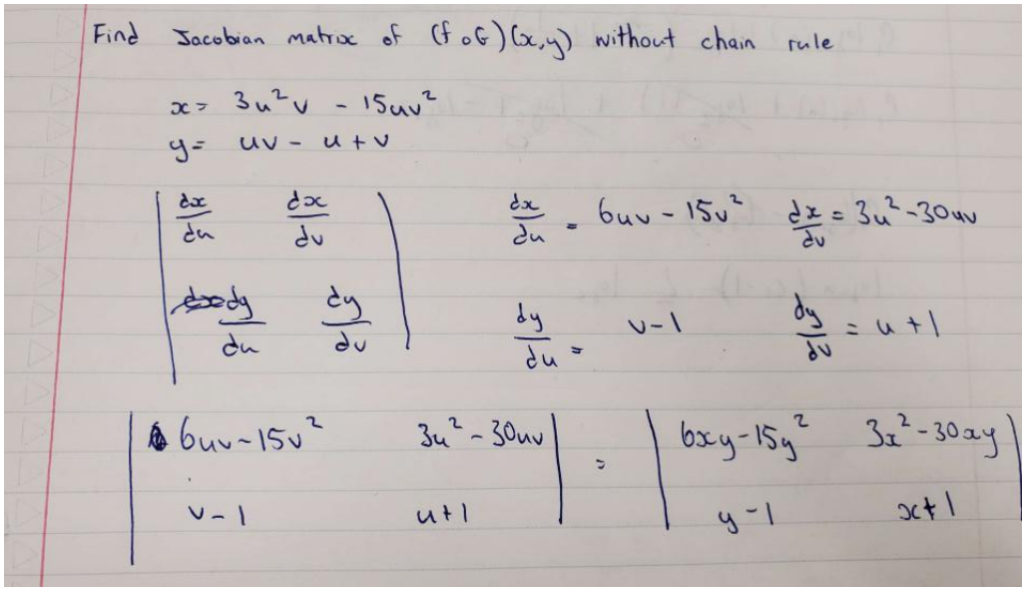
Assuming that G maps from R^m to R^n, and F maps from R^n to R^p, then:



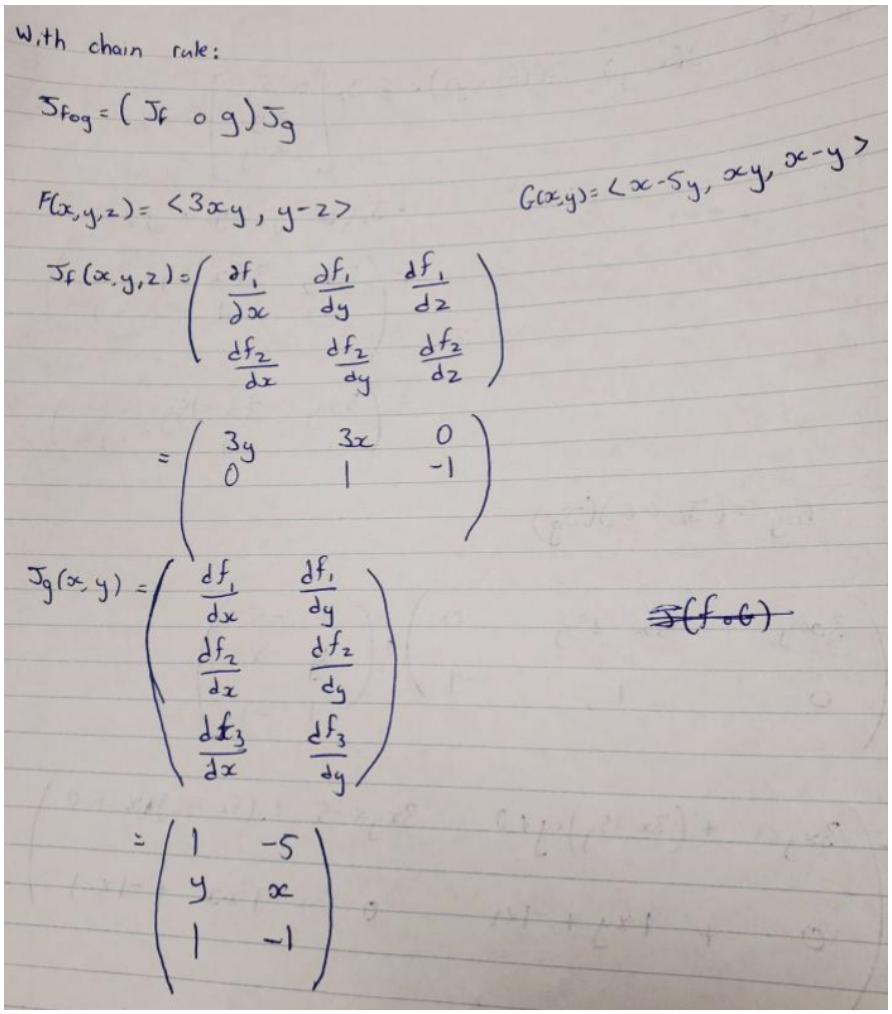
10. Composition of (F . G)(x,y):

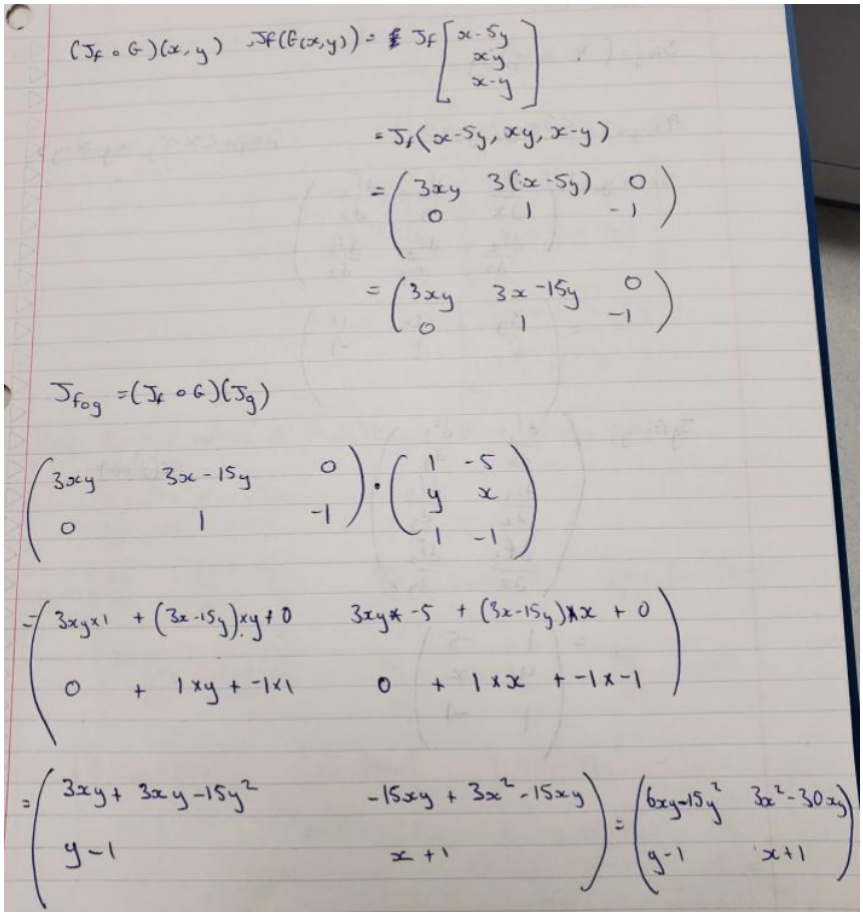


Find Jacobian matrix of the above composition without chain rule:



Find the Jacobian matrix using chain rule:



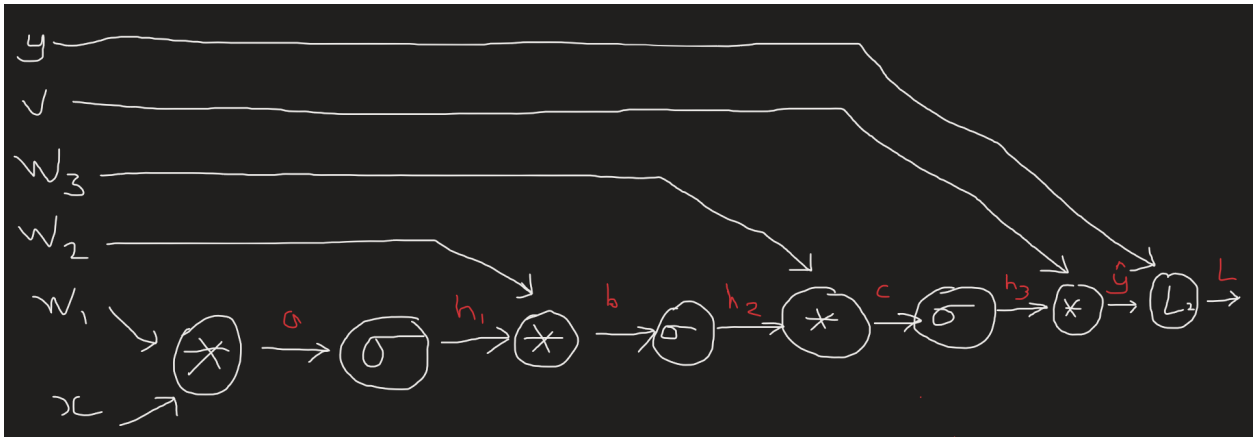


As we can see, we get the identical Jacobian matrix with and without chain rule

11. Explain the need to order nodes when traversing the computation graph in the forward and backward passes.

The nodes should be ordered so that it is possible to compute the output one after the other. There is functional dependencies between the nodes so if the nodes are not in order then the results can be affected. For example, in backward passes, the derivative of a node is calculated with respect to it's children nodes so it is important for the nodes to be in order so that the correct results can be obtained.

12. Scalar Computation Graph:
    a. Computation Graph:



    b.

X1 data input:

12) b For each data point, we compute the output of the hidden layer (z), and since a sigmoid activation is used, the following formula will be used: $z=(z_1, z_2, z_3)$

For data point (x_1, y_1):

$z_1 = \text{sigmoid}(w_{1_0} + w_1 \times x_1)$

$= \text{sigmoid}(0.01 + (0.02 \times 1) + (0.03 \times 2))$

$= 1/1 + e^{-0.09} = 0.522485$

$z_2 = \text{sigmoid}(w_{2_0} + w_2 \times x_2)$

$= \text{sigmoid}(0.03 + (0.01 \times 1) + (0.02 \times 2))$

$= 1/1 + e^{-0.08} = 0.519989$

$z_3 = \text{sigmoid}(w_{3_0} + w_3 \times x_1)$

$= \text{sigmoid}(0.02 + (0.03 \times 1) + (0.01 \times 2))$

$= 1/1 + e^{-0.07} = 0.517493$

Using $z=(z_1, z_2, z_3)$ as input for the output layer with linear activation, we can compute the prediction, $\hat{y}$:

$\hat{y}_1 = v_0 + (v_1 \times z)$

$= 0.01 + (0.02 \times 0.522485) + (0.03 \times 0.519989) + (0.04 \times 0.517493)$

$= 0.036749$

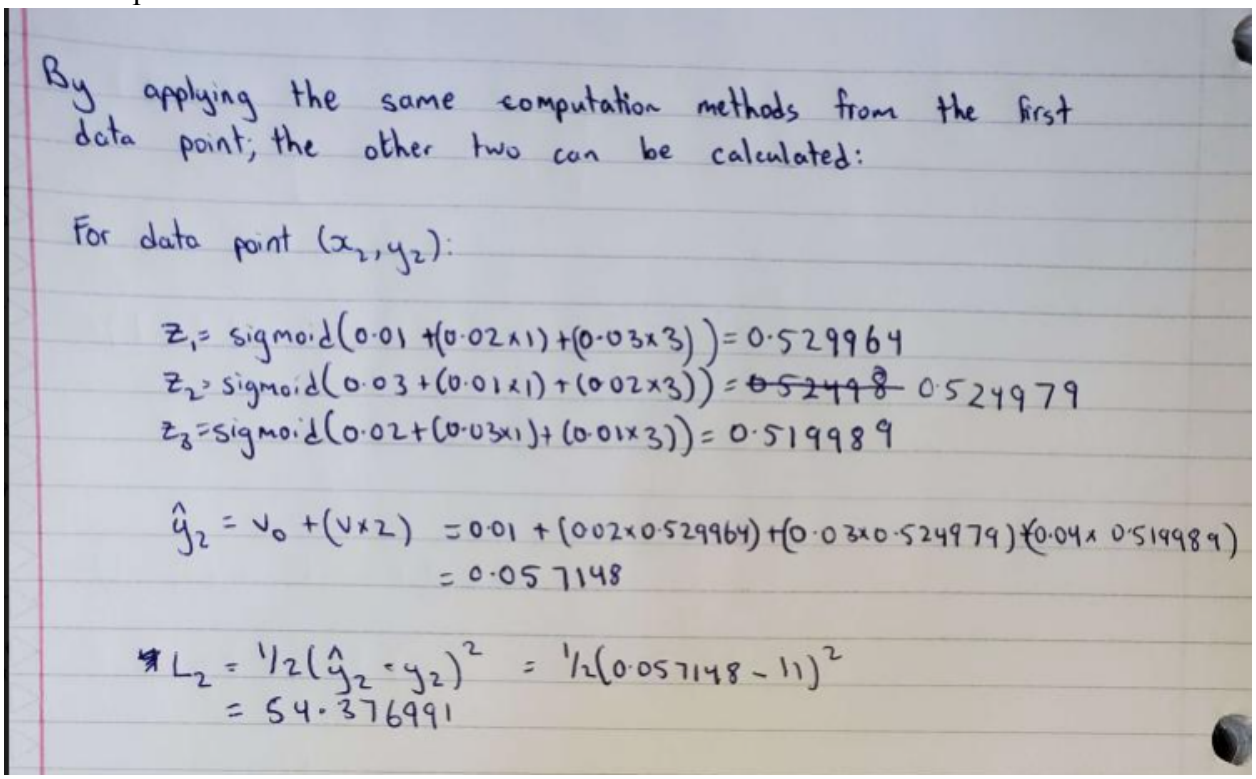
Using L_2 Loss for first data point we get:

$L_2 = 1/2(\hat{y}_1 - y_1)^2$

$= 1/2(0.036749 - 8)^2$

$= 31.55$

X2 data input:



By applying the same computation methods from the first data point; the other two can be calculated:

For data point (x_2, y_2):

$z_1 = \text{sigmoid}(0.01 + (0.02 \times 1) + (0.03 \times 3)) = 0.529964$

$z_2 = \text{sigmoid}(0.03 + (0.01 \times 1) + (0.02 \times 3)) = \cancel{0.52498}\ 0.524979$

$z_3 = \text{sigmoid}(0.02 + (0.03 \times 1) + (0.01 \times 3)) = 0.519989$

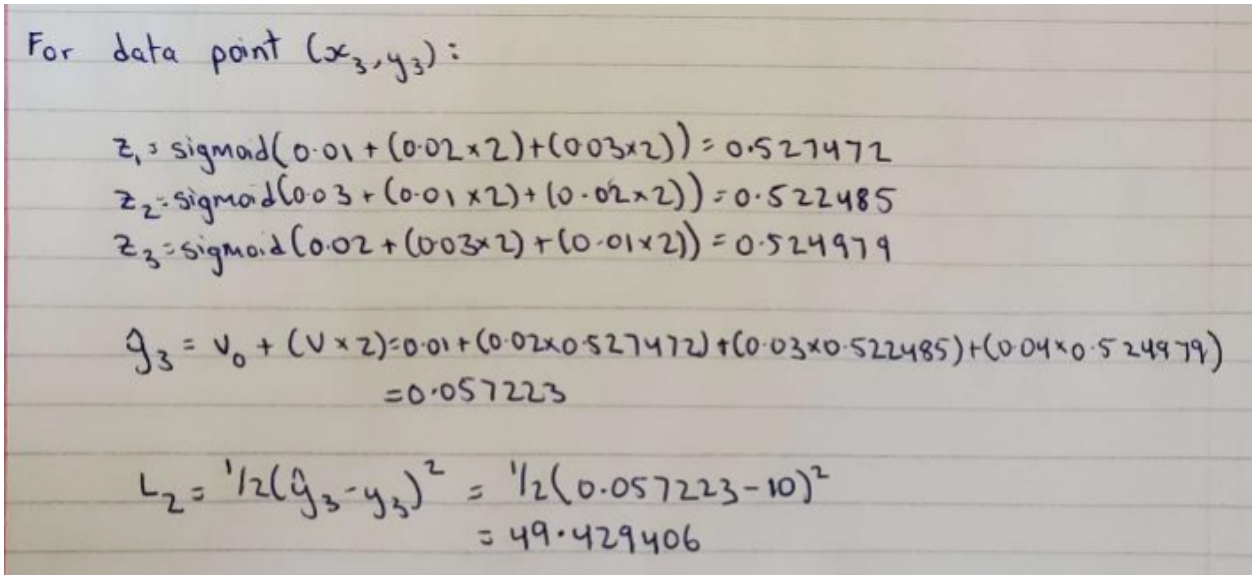
$\hat{y}_2 = v_0 + (V \times Z) = 0.01 + (0.02 \times 0.529964) + (0.03 \times 0.524979) + (0.04 \times 0.519989)$

$= 0.057148$

$L_2 = \frac{1}{2}(\hat{y}_2 - y_2)^2 = \frac{1}{2}(0.057148 - 11)^2$

$= 54.376991$

X3 data input:



For data point (x_3, y_3):

$z_1 = \text{sigmoid}(0.01 + (0.02 \times 2) + (0.03 \times 2)) = 0.527472$

$z_2 = \text{sigmoid}(0.03 + (0.01 \times 2) + (0.02 \times 2)) = 0.522485$

$z_3 = \text{sigmoid}(0.02 + (0.03 \times 2) + (0.01 \times 2)) = 0.524979$

$\hat{y}_3 = v_0 + (V \times Z) = 0.01 + (0.02 \times 0.527472) + (0.03 \times 0.522485) + (0.04 \times 0.524979)$

$= 0.057223$

$L_2 = \frac{1}{2}(\hat{y}_3 - y_3)^2 = \frac{1}{2}(0.057223 - 10)^2$

$= 49.429406$

c.

d.

13.

a. $\nabla f(x,y)$:

$$\nabla f(x,y) = \langle d/dx \ , \ d/dy \rangle$$

$$d/dx = 8x + 12y$$

$$d/dy = 18y + 12x$$

$$\nabla f(x,y) = \langle 8x + 12y \ , \ 12x + 18y \rangle$$

b. Compute the Jacobian matrix DF(1, 2)

$$x = u^2 + 2v$$

$$y = 3u + 4v^2$$

$$\begin{vmatrix} \frac{dx}{du} & \frac{dx}{dv} \\ \frac{dy}{du} & \frac{dy}{dv} \end{vmatrix}$$

$$\frac{dx}{du} = 2u \qquad \frac{dx}{dv} = 2$$

$$\frac{dy}{du} = 3 \qquad \frac{dy}{dv} = 8v$$

$$\text{Jacobian} = \begin{vmatrix} 2u & 2 \\ 3 & 8v \end{vmatrix} = \begin{vmatrix} 2x & 2 \\ 3 & 8y \end{vmatrix}$$

$$DF(1,2) = \begin{vmatrix} 2(1) & 2 \\ 3 & 8(2) \end{vmatrix} = \begin{vmatrix} 2 & 2 \\ 3 & 16 \end{vmatrix}$$

c. D(F . G)(2) without chain rule:

Find first (F . G)(x,y) = F(G(x)):

$$c) \quad (F \circ G)(x,y) = F(G(x,y)) = F\begin{bmatrix} x \\ x^2 \end{bmatrix}$$

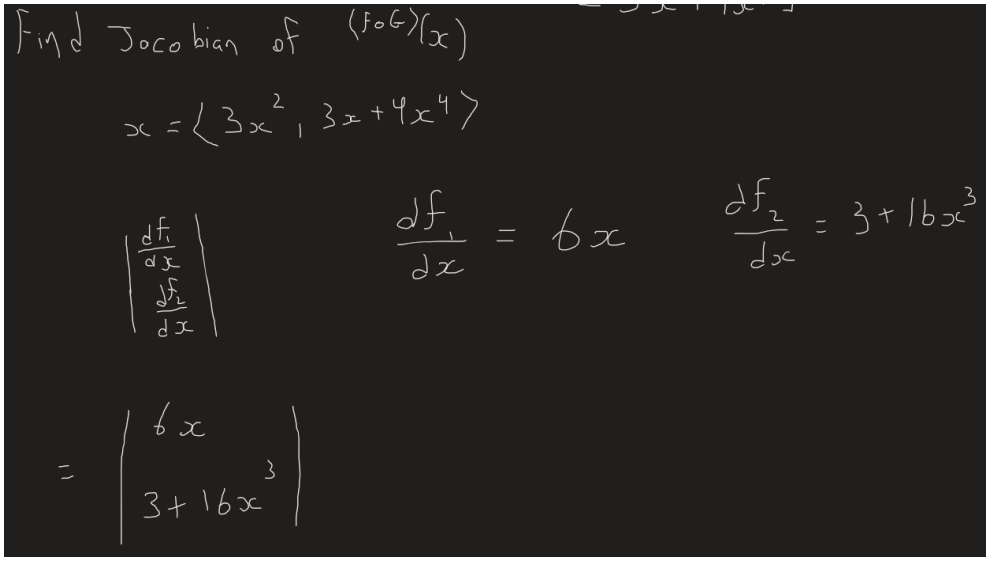
$$= F(x, x^2)$$

$$= \begin{pmatrix} x^2 + 2x^2 \\ 3x + 4(x^2)^2 \end{pmatrix}$$

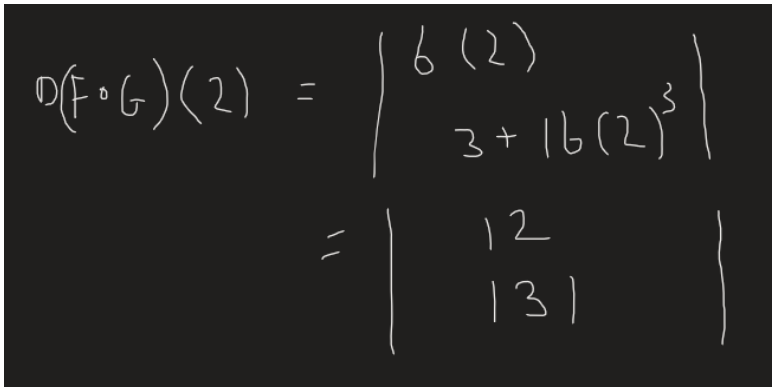
$$= \begin{bmatrix} x^2 + 2x^2 \\ 3x + 4x^4 \end{bmatrix}$$

$$= \begin{bmatrix} 3x^2 \\ 3x + 4x^4 \end{bmatrix}$$

Then find the Jacobian matrix of the above answer:



Lastly find the Jacobian matrix of D(F.G)(2) using the above answer:



d.

14. Assuming sigmoid activation function: