The Deep Learning Guide

Amelia N., Liesa L.

Last updated April 32, 2025

Contents

0 Introduction and Linear Algebra Stuff

0.1 Introduction

Categories of Machine Learning:

1. Predictive Supervised (classification and regression)
2. Descriptive unsupervised (clustering and association)
3. Reinforcement learning (self-fed inputs)

Classification

1. Linear
2. Non-linear

Represent almost anything as linear classification

1. Data representation can make a nonlinear system linear
2. Kernel methods
3. High dimensional representation
4. Feature spaces

Evolution of Learning

1. Rule based systems
   - Program → hand-designed program → output
2. Classic Machine Learning
   - Input → Hand-designed features → Mapping from features → Output
3. Representation Learning
   - Input → Features → Mapping from features → Output
   - Deep Learning: Input → Simple features → Additional layers of more abstract features → Mapping from features → Output

0.2 Math Review

The vector: $X = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}_{n \times 1}$

$X^T = \begin{bmatrix} x_1 & x_2 & ... & x_n \end{bmatrix}_{1 \times n}$

$|X| = \sqrt{\sum_1^n x_i^2}$

Property of transpose matrices: $(AB)^T = B^T A^T$

Vectors can be used for representing data or geometric information.

Normalization makes the vector unit length and preserves direction $\overline{X} = \dfrac{X}{|X|}$

Dot product $X \cdot Y = \sum_1^n x_i y_i$ also $U \cdot V = |V||U|cos(\theta)$ when θ is the smaller angle between them.

The cross product has magnitude $|U| = |X||Y|sin(\theta)$ is perpendicular to both original vectors and is computed by finding the determinant of this matrix $U = X \times Y = \begin{bmatrix} i & j & k \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} = (x_2y_1 - x_3y_2)i - (x_1y_3 - x_3y_1)j + (x_1y_2 - x_2y_1)k$

Determine two or more vectors' linear independence by row reducing their cumulative matrix $V_1...V_n \rightarrow \begin{bmatrix} V_1 & ... & V_n \end{bmatrix}$
If the final result has two or more rows that are multiples of each other, the system is linearly dependent. Otherwise, the system is linearly independent.

Orthogonal: dot product of two vectors is zero

Orthonormal: dot product is zero AND both vectors are unit vectors

Add matrices (they must have the same dimensions) $\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} a+x & b+y \\ c+z & d+w \end{bmatrix}$

Scale a matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times w = \begin{bmatrix} wa & wb \\ wc & wd \end{bmatrix}$

Multiply matrices $[m \times n] * [q \times p] = [m \times p]$ where $n = q$

Properties of determinants:

1. $det(AB) = det(A) * det(B)$
2. $det(AB) = det(BA)$
3. $det(A + B) \neq det(A) + det(B)$
4. If A is a diagonal matrix, then $det(A)$ is the product of its diagonal entries

Inverse matrices

1. The $det(A) \neq 0$
2. A must be square to have a determinant, and therefore must be square to have an inverse
3. $det(A^{-1}) = \dfrac{1}{det(A)}$
4. $(AB)^{-1} = B^{-1}A^{-1}$
5. $(A^T)^{-1} = (A^{-1})^T$
6. In the simplest case, a 2×2 matrix's inverse is $\dfrac{1}{det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

Rank
The rank of a matrix is equal to the number of pivots a matrix's row echelon form has.

0.3 Calculate Eigenvalues and Eigenvectors!

Given a matrix $\begin{bmatrix} 6 & 2 \\ 2 & 3 \end{bmatrix}$, calculate its Eigenvalues and Eigenvectors.

Eigenvalues: $det(A - \lambda I) = 0 \rightarrow det(\begin{bmatrix} 6-\lambda & 2 \\ 2 & 3-\lambda \end{bmatrix}) = 0$

$(6 - \lambda)(3 - \lambda) - 4 = 0 \rightarrow \lambda^2 - 9\lambda + 14 = 0 \rightarrow (\lambda - 7)(\lambda - 2) = 0$

$\lambda = 7, 2$

Then calculate the Eigenvectors by row reducing each matrix after plugging in all λs.

When $\lambda = 7$

$$\begin{bmatrix} 6-7 & 2 \\ 2 & 3-7 \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ 2 & -4 \end{bmatrix} \sim \begin{bmatrix} -1 & 2 \\ 1 & -2 \end{bmatrix} \sim \begin{bmatrix} 1 & -2 \\ 1 & -2 \end{bmatrix} \sim \begin{bmatrix} 1 & -2 \\ 0 & 0 \end{bmatrix}$$

$1x_1 - 2x_2 = 0 \rightarrow x_1 = 2x_2$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} 2x_2 \\ x_2 \end{bmatrix} \rightarrow x_2 \begin{bmatrix} 2 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

When $\lambda = 2$

$$\begin{bmatrix} 6-2 & 2 \\ 2 & 3-2 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \sim \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix} \sim \begin{bmatrix} 2 & 1 \\ 0 & 0 \end{bmatrix}$$

$2x_1 + 1x_2 = 0 \rightarrow x_1 = -\frac{1}{2}x_2$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} -\frac{1}{2}x_2 \\ x_2 \end{bmatrix} \rightarrow x_2 \begin{bmatrix} -\frac{1}{2} \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -\frac{1}{2} \\ 1 \end{bmatrix}$$

Diagonalization: $D = \begin{bmatrix} 7 & 0 \\ 0 & 2 \end{bmatrix}$ when $P = \begin{bmatrix} 2 & -\frac{1}{2} \\ 1 & 1 \end{bmatrix}$ with property $P^{-1}AP = D$

0.4 Singular Value Decomposition!

The goal of the SVD is to find the equation $A = U\Sigma V^T$ from an $m \times n$ matrix A. When encountering an SVD problem, do not panic. Instead, follow these steps to victory:

1. Determine V^T (This matrix will be of size $n \times n$)
   (a) Find A^TA
   (b) Find the eigenvectors of A^TA and normalize them $\frac{1}{\sqrt{\sum v_i^2}}v$
   (Divide each of their elements by the magnitude of the vector)
   $V = \begin{bmatrix} v_1 & ... & v_n \end{bmatrix}$
   (c) Determine V^T by taking the transpose of V
2. Determine Σ (This matrix will be of size $m \times n$)
   (a) Refer to your previous computations to determine $\sigma_{1...n}$. These values will be the square roots of each of the positive eigenvalues found in the previous step.
   (b) Build a diagonal matrix Σ from these values with all other entries being 0. Important note: the order of the columns of U corresponds with the order in which $\sigma_{1...n}$ are put into the matrix Σ.
3. Determine U (This matrix will be of size $m \times m$)
   (a) Calculate the vectors $u_{1...n}$ using $u_i = \frac{1}{\sigma_i}Av_i$ by using σ_i from step 2 and v_i from step 1.

(b) Build the matrix U from $u_{1...n}$

Property of SVD: $A^{-1} = VD^{-1}U^T$

The proof:

1. $A = UDV \longrightarrow AA^T = UDV^T(UDV^T)^T$
   $\longrightarrow AA^T = UDV^T(VD^TU^T)$
   $\longrightarrow AA^T = UDV^TVDU^T$
   $\longrightarrow AA^T = UD^2U^T$

   The columns of U are AA^T's Eigenvectors

2. $A = UDV \longrightarrow A^TA = (UDV^T)^TUDV^T$
   $\longrightarrow A^TA = (VD^TU^T)UDV^T$
   $\longrightarrow A^TA = VDU^TUDV^T$
   $\longrightarrow A^TA = VD^2V^T$

   The columns of U are A^TA's Eigenvectors

3. λ_i is an Eigenvalue of $A^TA \longrightarrow \sigma_i^2 = \lambda_i$

0.5 SVD Step by Step

Let's run through an example because this is difficult to comprehend by just staring at the step-by-step guide.

1. Let $A = \begin{bmatrix} 1 & -1 & 3 \\ 3 & 1 & 1 \end{bmatrix}$
   $A^TA = \begin{bmatrix} 1 & 3 \\ -1 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 3 \\ 3 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 2 & 6 \\ 2 & 2 & -2 \\ 6 & -2 & 10 \end{bmatrix}$
   Note how A^TA is symmetrical. Symmetric matrices are the easiest to work with.

2. Next, find the Eigenvalues and Eigenvectors of A^TA. Remember the characteristic polynomial: $(A - \lambda I)x = 0$ for the next couple of steps.
   $det\Big(\begin{bmatrix} 10-\lambda & 2 & 6 \\ 2 & 2-\lambda & -2 \\ 6 & -2 & 10-\lambda \end{bmatrix} \Big) = 0 \quad \longrightarrow \quad \lambda = 16, 6, 0$
   Always order the Eigenvalues from greatest to least.

3. Find the Eigenvectors by plugging the Eigenvalues back into the characteristic polynomial. Here's one of the Eigenvector computations:
   $\lambda_1 = 16 \begin{bmatrix} 10-16 & 2 & 6 \\ 2 & 2-16 & -2 \\ 6 & -2 & 10-16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
   $\rightarrow \quad \begin{bmatrix} -6 & 2 & 6 \\ 2 & -14 & -2 \\ 6 & -2 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
   Row reduce the matrix $\begin{bmatrix} -6 & 2 & 6 & 0 \\ 2 & -14 & -2 & 0 \\ 6 & -2 & -6 & 0 \end{bmatrix}$ to get $\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
   Get the equations $x_1 - x_3 = 0$ and $x_2 = 0$ from the row reduced matrix.

Construct the final Eigenvector $v_1 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} x_3 \\ 0 \\ x_3 \end{bmatrix} \rightarrow x_3 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$

Find the unit vector: $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}$

Repeat the process for the other Eigenvalues, get V, and then derive V^T.

Final Eigenvector $V = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{6}} \\ 0 & -\frac{1}{\sqrt{3}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{6}} \end{bmatrix}$, $V^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \end{bmatrix}$

4. Take the roots of the positive Eigenvalues and put them in the diagonals of the Σ matrix. Fill all other slots with zeroes.

   $\Sigma = \begin{bmatrix} \sqrt{16} & 0 & 0 \\ 0 & \sqrt{6} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 0 & 0 \\ 0 & \sqrt{6} & 0 \end{bmatrix}$

5. You're almost there. Time to find U.

   Here's an example computation for the first rooted Eigenvalue σ_1.

   $$Av_1 = \sigma_1 u_1 \rightarrow u_1 = \frac{1}{\sigma_1} Av_1 = \frac{1}{4} \begin{bmatrix} 1 & -1 & 3 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

   Repeat for σ_2

   $$u_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

   Combine the results into

   $$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

   The final decomposition:

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 \\ 0 & \sqrt{6} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \end{bmatrix}$$

Note how the matrices would still be compatible if there were three σs as U would be 2×3 and $\Sigma \times V^T$ would be 3×3, which are compatible for matrix multiplication.

0.6 Systems of Linear Equations

Systems of linear equations can be solved manually, but this is time consuming. To quickly solve a system of linear equations, the echelon form of a matrix can be used.
A matrix can be built from the equations

$$\begin{matrix} 3x - 4x^2 = 9 \\ 6x + 5x^2 = 2 \end{matrix} \rightarrow \begin{bmatrix} 3 & -4 & 9 \\ 6 & 5 & 2 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & \approx 1.36 \\ 0 & 1 & \approx -1.23 \end{bmatrix}$$

Good to know: Systems that have more equations than variables are called over-determined systems. Systems that have fewer equations than variables are called under-determined systems, can always be solved, will always have infinitely many solutions when consistent, and usually have free variables.

1 Image Classification

1.1 Image Classification

Let's classify images. Here are some variances that may be on the exam:

1. Viewpoint
2. Illumination
3. Deformations
4. Clutter
5. Occlusion
6. Intraclass Variations

1.2 Drawing a Line

After a model has classified a dataset, the big question is what distance metric should be used to totally split that sucker up! There are two options: L1 distance and L2 distance.

1. L1 "Manhattan" Distance

   $d_{Mntn}(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

   (a) Angular; works on a grid
   (b) Often used for higher dimensional data
   (c) Efficient as square root operation is computationally expensive

2. L2 "Euclidean" Distance

   $d_{Eucl}(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$

   (a) Abstract; works continuously
   (b) Is more accurate for the most accurate shortest distance between two points
   (c) More computationally expensive as the square root operator is used

1.3 Classification in a Machine Learning Context

Training a network on an entire dataset can lead to the model being extremely accurate for that dataset specifically, which can lead to good results under false pretenses. To combat this, the data can be split up strategically during the machine learning process to find the best model when its results are compared to unknown data.

Before using any strategy to train a network, it's important to separate some of that data to use as test data. This data should not be touched at all during the training process. Now for the data using during training. Since it's computationally expensive to run every single piece of data through the network at once, the data is split up into various folds which will then be used in sequence to improve the loss of the network. For example, the first Fold will receive random weights and biases and run a number of passes using its data. The network then passes on the current progress to the next fold. This ensures that all the data is used and that the process isn't as computationally expensive as it would be if there were just one pass with all the data.

The final network can then be demonstrated by using data that the network has never seen before (the Test Data) to demonstrate its accuracy candidly.

Trivia:

1. What is a significant challenge in image classification? The semantic gap
2. How does nearest neighbor training work when building a classifier? Memorize all data and labels, then predict the label by finding the most similar training image
3. What is an issue with nearest neighbor training? It takes up a lot of space
4. What three categories should you split your data into when building a model? Training, validation, and test. Choose parameters that work best on validation and evaluate on test.
5. Why is it bad to split your dataset into training and test only, choosing parameters that work best on test? You can't predict how it the model performs on new data
6. What happens in cross-validation? First split your dataset into folds. Try each fold as validation and average the results.
7. What is the parametric approach to classification? Take image features, then map features to a class score using a linear function
8. How can we tell if the results from a model training are good? Use a loss function
9. What are the next steps in model training after using a loss function? Optimize W and b to minimize loss. This consists of backprop (to compute gradients), then gradient descent (to update weights)
10. What values do W and b start with? Random values

2 Loss Function

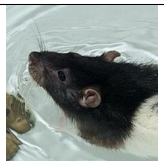
2.1 Loss Table

Loss can be used to reward or punish a neural network during training.

$$\text{Loss function: } \sum_{j \neq y_i} \begin{cases} 0 & \leftarrow s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \leftarrow \text{otherwise} \end{cases}$$

Observations about loss:

1. Changing the individual scores a little doesn't impact the final loss that much

			
Catness	**2.1**	1.4	2.3
Ratness	3.5	**5**	2
Flowerness	-2.3	2.8	**-5**
Loss	2.4	0	16.3
Final Loss	9.35		

2. The bounds of the loss are $[0, \infty)$
3. If W (weights) are initialized to small numbers, the loss will then be $N - 1$ where N is equal to the number classes
4. The loss can be averaged in a variety of different ways, but they still reflect the same reality
5. There are infinitely many ways of achieving zero loss

2.2 Regularization

When training the model on a certain dataset, it may be rewarded too well within its limited dataset. To counteract this, a regularization function is used to prevent the model from excelling while comparing its results to the training data.

$$L(W) = \frac{1}{N} \sum_1^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

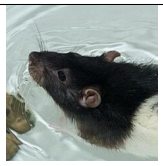
Where λ is the strength of the regularization.

2.3 Softmax Loss

Softmax Loss is a way of making the data easier to read and quantify in one sweep. When applying Softmax Loss, the first step is to convert the values in each column associated with the class of an item to a percentage. This is called the normalized probability of the class. This makes the values easier to work with, but most importantly, relative to each other. Knowing one value as a percentage is infinitely more valuable to an understanding of how the network has classified that image compared to knowing one arbitrary, unscaled value without context. Even with context, it can be hard to quantify the impact of arbitrary values (as seen above in the beginning of the loss section) on the final loss.

The question is how the Minus Log-Probability is calculated. Only the natural log $ln(P)$ of the value of the desired class in a certain column has to be calculated. For example, only the normalized probability of the Catness of the cat is required to calculate the MLP (Minus Log-Probability) which comes out to 1.61. At first glance, this may make no sense. How are the other values accounted for if only the normalized probablity of the Catness is being plugged in? Well, that's because the percentage is *inherently* relative to the other values. All the values went into calculating each of the parts of the percentage. As seen in the chart, all the columns add to one. They are relative to each other.

When you find the relative loss to the incorrect classes, you can reward models that have a value approaching zero. This is because, in theory, the value of the classifications should come out to, say, $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$. This would mean that a model is 100% confident that whatever it's classifying is the correct class (assuming that the correct class is in fact the first entry), resulting in $-ln(1) = 0$, or zero loss!

			
Catness	**0.2**	0.02	0.569612
Ratness	0.8	**0.88**	0.43
Flowerness	0	0.1	**0.000388**
Minus Log-Probability	1.61	0.129	7.85
Final Loss	3.198		

$$\text{Softmax function: } P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$
$$\text{Minus Log-Probability Function: } L_i = -ln(P)$$

Reference the table at the beginning of the section and note how the flower's classification is the least accurate at -5. The Flowerness as a normalized probability comes out to 0.000388, which ends up rendering a MLP of 7.85, almost 5 times worse than the cat's MLP.

Softmax trivia:

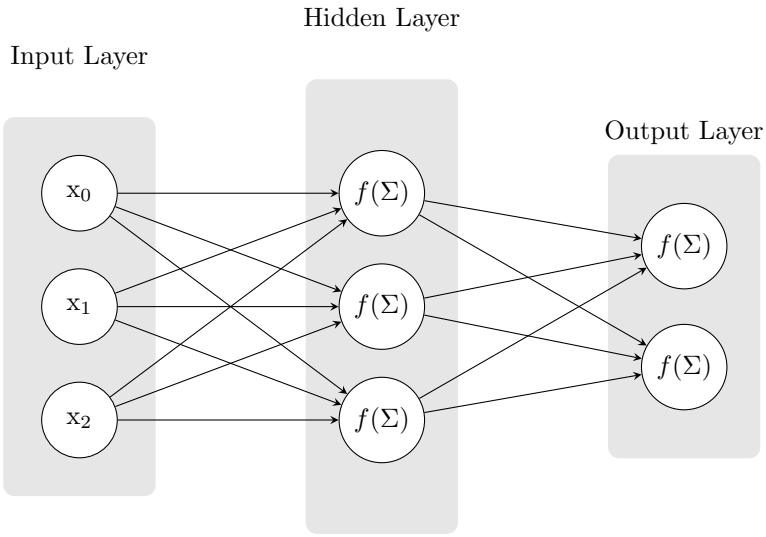
1. The bounds of $L_i : [0, \infty)$
2. L_i is equal to $ln(N)$ at initialization where N is the number of classes

Trivia:

1. What is the equation for Hinge Loss? $L_i = \sum max(0, s_j - s_y + 1)$
2. If you have N number of classification labels, what do you do to get total loss? Sum the loss of each category and divide by N. (i.e., get the average)
3. Given W_1 = [1 0 0 0] and W_2 =[0.25 0.25 0.25 0.25], what does L2 regularization prefer? W_2 because L2 prefers weights to be spread.
4. Given W_1 and W_2 above, what does L1 regularization prefer? W_1 because L1 prefers weights to be concentrated.
5. What happens without regularization? Model fits too closely to training data, and you get large loss values for test data.
6. The purpose of the Softmax function is to maximize the probability of what? The correct class

3 Neural Networks

A neural network consists of various layers that are composed of nodes. The first layer in an input layer, which consists of the data that's currently getting classified. The last layer is the output layer. This is the layer associated with the different labels associated with the different classes that can result from the inputted data. The layers in between are called hidden layers. These hidden layers are responsible for applying weights, biases, and an activation function to the previous inputs that were fed into the layer. The output layer may or may not have weights, biases, and/or an activation function.



3.1 Forward Pass

1. Receive Inputs: x_1, x_2, x_3
2. Multiply Inputs by Weights: Compute w_1x_1, w_2x_2, w_3x_3
3. Compute Weighted Sum: $z = w_1x_1 + w_2x_2 + w_3x_3 + b$
4. Apply Sigmoid Activation Function: $y = \dfrac{1}{1+e^{-z}}$
5. Compute Loss*: L

This process is pretty straightforward. The inputs get multiplied by their corresponding weights, the bias is added, and then the result is put through the activation function. The only step that only is executed sometimes is the computation of the loss. The loss of the network only should be computed for the output layer. Once the forward pass for the entire network has been run through, the backwards pass comes next.

3.2 Backwards Pass

The goal of the backwards pass is to compute the final gradient of the network as a whole. To do this manually, the chain rule can be applied to the operations that were performed on the values to get individual gradient matrices for each layer which can be multiplied by a small step factor and then applied to the original weight matrices that they correspond to. The gradient matrix of a layer must have compatible dimensions with its corresponding weight matrix.

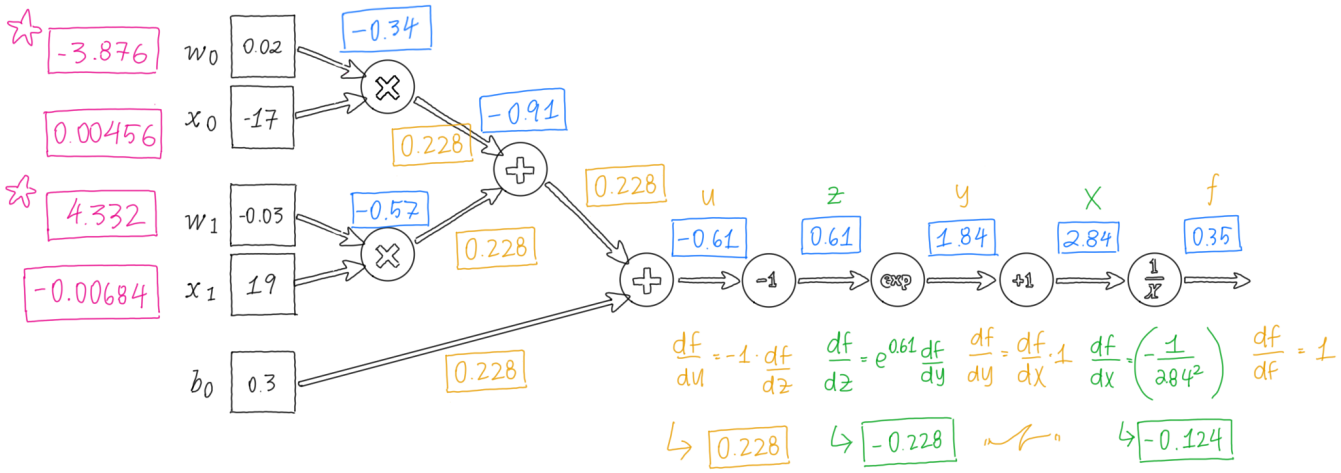
Assuming that the activation function applied on the layer was a sigmoid function ($\dfrac{1}{1+e^{-z}}$), this is what the chain rule will look like.

1. $\dfrac{df}{df} = 1$

   The derivative with respect to itself is one.

2. $\dfrac{df}{dx} = \dfrac{df}{dx}\dfrac{df}{df} = \Big(-\dfrac{1}{x^2}\Big) \times 1$

   The derivative of the function applied is taken.



3. $\frac{df}{dy} = \frac{dx}{dy}\frac{df}{dx} = 1 \times \frac{df}{dx}$

   The operation applied on the original number $1 + y$ has a derivative of 1.

4. $\frac{df}{dz} = \frac{dy}{dz}\frac{df}{dy} = e^{prev} \times \frac{df}{dy}$

   The derivative of e^x is e^x.

5. $\frac{df}{du} = \frac{dz}{du}\frac{df}{dz} = -1 \times \frac{df}{dz}$

   The derivative of the operation $-u$ performed on the value is -1.

6. All $+$ addition operations performed on a number will derive from an equation that looks like $u(w) = w + c$. The derivative of these equations will always be 1.

7. The final operations performed during the backwards pass are the ones that are for the original multiplications between the weights and inputs. Take the weight w_0 and input x_0. The formula $f(w_0) = w_0 x_0$ and $f(x_0) = w_0 x_0$ both each result in the same value but are that value with respect to a different input. Taking each of their derivatives, it would mean that their respective variable would derive to 1 and the opposite value (now a multiplying constant) will remain. This is why the 'current chain rule value' is seemingly multiplied by the opposite value - it's just treating the opposite value as a constant and the respective value as the variable.

   To reiterate because this can be confusing at first, the same process of multiplying the previous chain rule value is by the current derivative is applied here. Let's find the derivative of $f(w_0) = w_0 x_0$ to get the value that will correspond to its final gradient value. $\frac{df}{dw_0} = 1 \times x_0$. Observe how x_0 remains because it's treated as a constant. It's like taking the derivative of $2x$ with respect to x. $\frac{df}{dx} 2x = 2$. The w_0 is treated the same as the 2 in this situation.

3.3 Sidenote: Neuron Activation Functions

There are a variety of activation functions to choose from. These include the

1. Sigmoid

   $\sigma(x) = \frac{1}{1+e^{-x}}$

   - Often used for when we want to predict the probability as an output (since sigmoid squashes output to probabilty between 0 and 1)
   - Commonly used in output layer for binary classification

2. Leaky ReLU

   $max(0.1x, x)$

   - Outputs input directly if positive, and 0.01 for negative inputs
   - Good for deep networks
   - Addresses "dying ReLU" problem by allowing some degree of negative inputs.

3. Tanh

   $tanh(x)$

   - Outputs values between -1 and 1
   - Commonly used in hidden layers of a neural network because its zero-centered nature can make training more efficient than sigmoid

4. Maxout

   $max(w_1^T x + b_1, w_2^T x + b_2)$

   - Returns the max of the inputs
   - A generalization of the ReLU and leaky ReLU

5. ReLU

   $max(0, x)$

   - Outputs input directly if it's positive, and 0 otherwise
   - Good for shallow (few hidden layers) networks
   - Can suffer from "dying ReLU" problems, where neurons become inactive and stop learning if their input is negative
   - Less computationally expensive than tanh and sigmoid

6. ELU

   $$\begin{cases} x & \leftarrow x \geq 0 \\ a(e^x - 1) & \leftarrow x < 0 \end{cases}$$

   - "Exponential Linear Unit"
   - Provides smoother curve than ReLU, which can lead to more stable gradient flow during back prop
   - More computationally expensive than ReLU
   - Can produce negative outputs

3.4 Applying the Results

1. Update Weights: $w_i = w_i - \eta \frac{\partial L}{\partial w_i}$
2. Update Bias: $b = b - \eta \frac{\partial L}{\partial b}$
3. Repeat from Forward Pass

When updating the weights and biases, the final gradient is multiplied by a small constant η so that only small steps are taken down the final gradient descent so that no overshoot occurs.

3.5 Backpropagation Using the Jacobian

Instead of doing every computation manually, it's easier to just use matrix multiplication to get the job done. Given the values $W = \begin{bmatrix} 0.9 & 0.2 \\ 1.5 & -2.6 \end{bmatrix}, x = \begin{bmatrix} 0.19 \\ 0.83 \end{bmatrix}$ and the activation function $f(x; W) = ||W \cdot x||^2 = \sum (w_i \cdot x)^2$. Note: the bars are not absolute value bars, but the vector 2-norm. Execute the forward pass by doing the matrix multiplication $Wx = \begin{bmatrix} 0.337 \\ -1.873 \end{bmatrix}$ and then activate the result to get 3.62. The result ends up being a single number in this case because the activation function is a cumulative sum of squares. The partial, relative derivatives come out to

$\nabla_r f = 2r$ where $r = Wx$ so $\nabla_r f = \begin{bmatrix} 0.674 \\ -3.746 \end{bmatrix}$

$\nabla_w f = \nabla_r f \cdot x^T = \begin{bmatrix} 0.128 & 0.559 \\ -0.711 & -3.109 \end{bmatrix}$

$\nabla_x f = W^T \cdot \nabla_r f = \begin{bmatrix} -5.0124 \\ 9.8744 \end{bmatrix}$

Done like dinner! Not every problem will use this activation function, though, so it's important to understand *why* $\nabla_r f = 2r$. In this case, deriving $(Wx)^2$ got there, but in an ReLU problem, the piecewise aspect about the function must be accounted for in backpropagation (there's no one way of doing it, the derivative will be based on the output).

Trivia

1. The derivative of the sigmoid function is what? The sigmoid function multiplied by one minus itself. $\frac{d\sigma(u)}{du}$ = (1-$\sigma(u)$)$\sigma(u)$

4 Convolutional Neural Networks

4.1 Handling Multi-Layer Networks

Neural networks can be two layered (one hidden layer) or multi-layered (two or more hidden layers). The issue with having a deeper neural network is the fact that they are inefficient as the number of neural connections they have increase rapidly. However, if connections are limited, nodes in further in the network have almost, if not the same amount of much visibility further back into the network. The nodes early in the network that affect a node later in the network make up the later node's receptive field. As receptive fields of deeper nodes aren't affected when limiting the number of connections that each node has, the network can save on computational power while retaining its computational integrity.

4.2 Convolutional Neural Networks

Convolutional neural networks exist to achieve one goal: increasing the scope of the network while reducing the number of computations that network must execute to get a final result. By convoluting the data, the

network can manage fewer parameters and therefore perform fewer operations for a final result. By stacking convolved networks and padding the results to maintain the size of the original image, the network can chain these convolutional layers to increase depth while not getting ridiculously expensive to train and process data with.

A convolutional layer takes an input image and outputs an altered image with sizing according to the equation listed below. What it's really doing is filtering data from the original image into an output image. The output image is usually smaller but appropriate padding during the process can preserve the original image size.

1. Convolve a smaller kernel with sections of the input image
2. Filter the green pixels with a second convolution kernel
3. Filter the red pixels with a third convolution kernel
4. Continue chaining convolution layers and activation maps

   The deeper the layer, the higher level the features seen in that layer will be.

Trivia:

1. For a 7x7 input with a 3x3 kernel (assuming a stride of 1), what will the size of the output be?

   $(7-3+1\times 7-3+1)=5\times 5$

   See general formula below.

2. For the same premise but with a stride of 2 instead of 1, what will the size of the output be?

   3x3.

   General formula: $dim_{out}=\lfloor\frac{N-F}{S}+1\rfloor$ where N is the width and height of the input, F is the width and height of the kernel, and S is the stride size.

   Formula to pad the output to be the same size as the input image: $P=\frac{[(S-1)\times N]+(F-S)}{2}$ where $S>1$ else $P=\frac{F-1}{2}$

3. Input: $32\times 32\times 3$
   Kernel: 10 filters 5×5, stride of 1, pad 2

   $dim_{out}=\frac{N+2P-F}{S}+1=\frac{32+4-5}{1}+1=32$, 10 filters, Final Output: $32\times 32\times 10$

   Number of parameters: $5\times 5\times 3=75+1$ bias per filter
   Total parameters: 10 filters $\times$ 76 parameters per filter = 760 parameters