

Multi-Variable Calculus

|  |  |
| --- | --- |
| Assignment 1 | Applications of Multi-Variable Calculus in Computer Science |
| Name | Muhammad Abdullah |
| Sap ID | 53457 |
| Section | BSCS-3B |
| Instructor | Dr. Maria Akram |

# Optimization in Machine Learning:

Optimization in machine learning is a process of finding the best model parameters by minimizing or maximizing objective functions, such as loss functions. A common objective in supervised learning is to minimize a loss function, which measures the discrepancy between predictions and true labels. For example, in linear regression, the loss function is the Mean Squared Error (MSE):

[1]

where:

* Yi​ is the true label for the iii-th data point,
* ​ is the predicted value for the iii-th data point,
* N is the total number of data points,
* w and b are the parameters of the model (weights and bias).

To minimize the loss function, Gradient Descent, a widely used optimization technique, is employed. It updates the parameters iteratively using the gradients of the loss function. The equation of loss function is given as,

[1]

[1]

where:

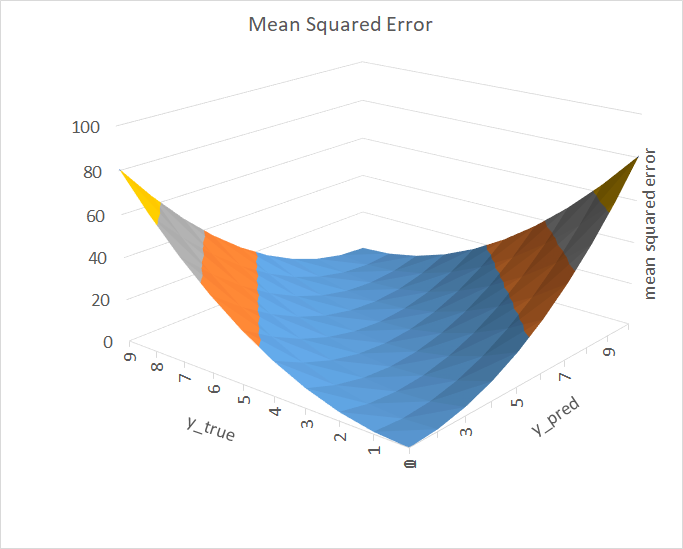
* ,​ are the gradients of the loss function for w and b,
* η is the learning rate, which controls the step size.

The gradients for the MSE loss function are calculated as follows:

[1]

[1]

The following figure shows the graph for mean squared error with respect to y\_true and y\_predict.



Here is the code for above graph,

|  |
| --- |
| **def mean\_squared\_error(actual, predicted):**  **sum\_square\_error = 0.0**  **for i in range(len(actual)):**  **sum\_square\_error += (actual[i] - predicted[i]) \*\* 2.0**  **mean\_square\_error = 1.0 / len(actual) \* sum\_square\_error**  **return mean\_square\_error**    **y\_true = [3, -0.5, 2, 7]**  **y\_pred = [2.5, 0.0, 2, 8]**    **mse1 = mean\_squared\_error(y\_true, y\_pred)**  **print(f"MSE = {mse1}")**  **# MSE = 0.375**      **from sklearn.metrics import mean\_squared\_error**  **mse2 = mean\_squared\_error(y\_true, y\_pred)**  **print(f"MSE = {mse2}")**  **# MSE = 0.375** |

Here, xi​ is the input feature for the i-th data point. By iteratively updating w and b, the loss function L(w,b) is minimized, leading to improved model predictions. Advanced optimization methods such as Stochastic Gradient Descent (SGD), Adam, and RMSprop refine these principles to handle large datasets, adaptive learning rates, and momentum for faster convergence. Visualizations, such as contour plots of the loss surface, illustrate the trajectory of optimization algorithms as they converge towards minima.

# Computer Vision With Sobel Masks:

Computer Vision and Image Processing are essential applications of multivariable calculus, particularly in techniques such as edge detection and image enhancement. These techniques rely on calculating the rate of intensity changes in images, which involves partial derivatives. The Sobel filter, a widely used operator in edge detection, is a prominent example of this application.

Mathematically, the Sobel filter computes the partial derivatives of the intensity function I(x,y), representing the image's pixel intensities. The horizontal gradient Gx ​ is calculated as:

[2]

Likewise the vertical gradient Gy is given as

[2]

The magnitude of the gradient, which determines the intensity of edges, is given by combining these partial derivatives:

[2]

To enhance edges and highlight changes in intensity, convolution operations are performed using kernels that approximate these derivatives. For instance, the Sobel operator uses the following kernels to calculate Gx­ and Gy,

[2]

[2]

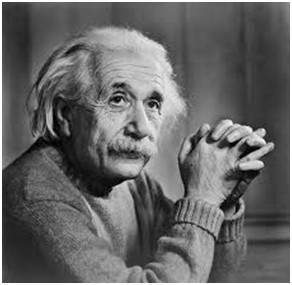
These kernels are convolved with the image matrix I(x,y), producing gradients highlighting the edges. The output images from these operations showcase the regions of rapid intensity change, which correspond to edges in the image.

Let’s consider an example and apply Sobel filter.

sobel operator is also used to detect two kinds of edges in an image:

* Vertical Direction
* Horizontal Direction

in sobel operator the coefficients of masks are not fixed and they can be adjusted according to our requirement unless they do not violate any property of derivative masks. Both horizontal and vertical masks or kernel is given in equations (9) and (10) respectively. We will apply Sobel masks on following image



Firstly, let’s see horizontal masks. This mask will prominent the horizontal edges in an image. It also works on the principle of above mask and calculates difference among the pixel intensities of a particular edge. As the center row of mask is consist of zeros so it does not include the original values of edge in the image but rather it calculate the difference of above and below pixel intensities of the particular edge. Thus increasing the sudden change of intensities and making the edge more visible. After applying the horizontal masks we will have the image resulting in following image:



Secondly, let’s see vertical masks. When we apply this mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in a edge region.

As the center column is of zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also the center values of both the first and third column is 2 and -2 respectively.

This give more weight age to the pixel values around the edge region. This increase the edge intensity and it become enhanced comparatively to the original image. Now we will have our resulting image as follows:

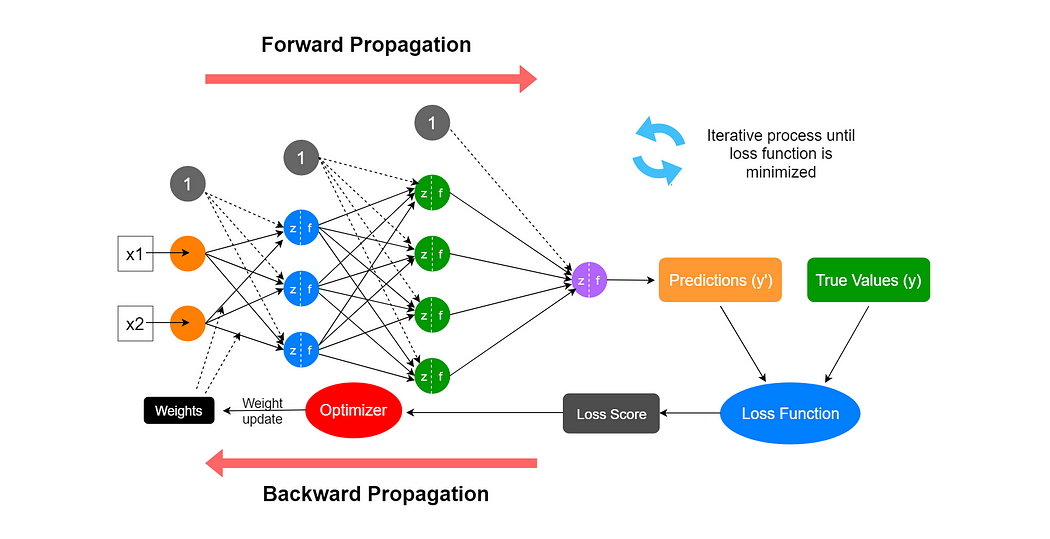


# Neural Networks and Back Propogation:

Neural Networks and Backpropagation play a fundamental role in training deep learning models, relying heavily on multivariable calculus. The backpropagation algorithm is used to compute gradients of the loss function with respect to the weights in the network, which are then updated to minimize the loss function. This optimization process is driven by the chain rule from multivariable calculus.

Consider a neural network where the loss function is represented as L, the output of a neuron as y, the input to the neuron as z, and the weight as w. To compute the gradient of the loss function ∂L/∂w​, the chain rule is applied as follows:

[1]

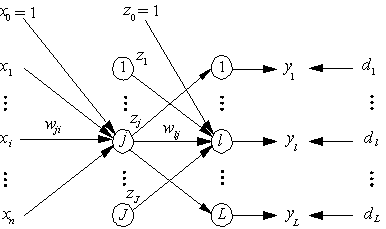


Here:

* ∂L/∂y ​ measures how the loss changes with the neuron's output,
* ∂y/∂z​ measures how the neuron's activation changes with its input, and
* ∂z/∂w​ measures how the input changes with the weight.

Each weight in the network is updated iteratively using the gradient descent algorithm:

[1]



where η is the learning rate that controls the step size of updates.

During training, these computations are performed for all layers in the network, starting from the output layer and propagating backward through the hidden layers to the input layer. This process ensures that the network learns to reduce the error in predictions by adjusting the weights effectively.

# Data Visualization:

Data visualization is a powerful tool for analyzing and understanding multivariabledata. Techniques such as contour maps, heat maps, and 3D surface plots make it possible to graphically represent relationships between variables and gradients.

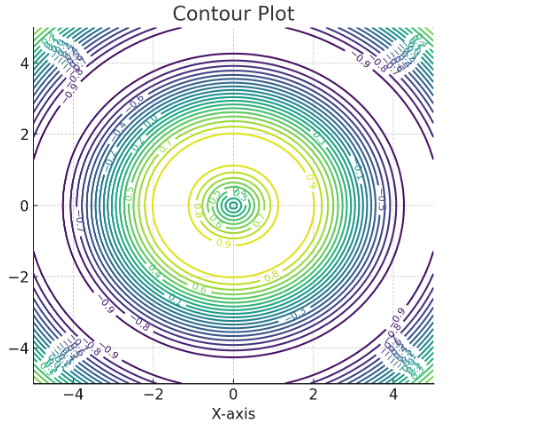
For a multivariable function z=f(x,y), visualization techniques are used to depict the dependency of z on x and y. One of the key aspects is calculating the gradient field, which is expressed as:

Let’s chose a function and find gradients with respect to x and y and then plot the function as a surface plot and contour plot.

Gradient with respect to x:

Gradient with respect to y:

Plot of the above function:



With slopes and gradients we can find the trends of the industry depending upon the situation.

# Game Physics:

Game physics and simulation play a critical role in creating immersive and realistic game environments. Multivariable calculus is extensively utilized to model physical phenomena such as particle motion, fluid dynamics, and collision detection, which are fundamental to modern game development. These simulations rely on mathematical models that describe the behaviour of objects under various forces and interactions.

The Navier-Stokes equations, which describe the motion of fluid substances, are given by:

The following image contains a simulation based upon Navier-Stokes equation



Here:

* u = velocity field of the fluid (a function of space and time)
* p = pressure field of the fluid
* = density of the fluid
* = kinematic viscosity
* = external forces acting on the fluid (e.g, gravity)
* Temporal change of velocity represents how the velocity of the fluid changes with time.
* Convective term accounts for the change in velocity due to the movement of fluid within the field
* Presseure gradient term represents the effect of pressure
* Viscous term models the internal friction within the fluid, leading to energy dissipation
* External forces includes influences like gravity or user-controlled forces in a game

The equations are solved numerically in game engines to simulate fluid flow, smoke, fire, and other dynamic systems. applications include fluid simulation for water, smoke, and explosions, collision detection to determine when and where objects in a game world collide using calculus-based approaches to compute trajectories and impact points, and particle systems for effects like fire, rain, or magic spells.

# Word Embedding in NLP:

Natural Language Processing (NLP) is a critical area in artificial intelligence where multivariable calculus plays a significant role in training models for tasks like word embeddings. Word embeddings such as Word2Vec are trained using optimization techniques, where multivariable calculus is employed to compute gradients and optimize objective functions. These embeddings represent words as vectors in a high-dimensional space, capturing semantic relationships between them.

The training of Word2Vec involves optimizing an objective function. One commonly used objective function for embeddings is:

Here:

* = Objective function to minimize (negative log-likelihood)
* N = number of training samples (word context-pairs)
* = Context word associated with the target word wi
* = Target word
* = Model Parameters (word vectors and biases)
* is given as

Here:

* = Vector representation of the target word wi
* = Vector representation of the context word ci
* V = size of the vacobulary

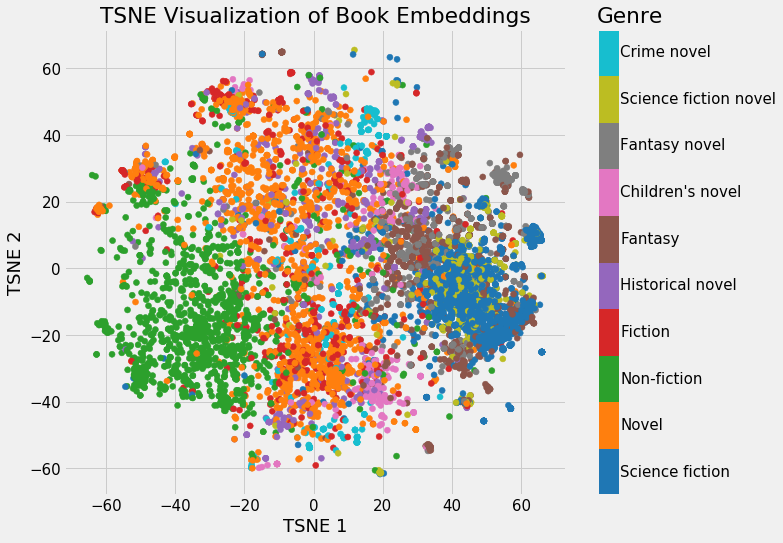
The optimization process typically uses algorithms like Stochastic Gradient Descent (SGD) to minimize J(θ), refining the embeddings iteratively.

Word embeddings trained through this process capture semantic meanings, where similar words are placed closer together in the vector space. For example, words like "king" and "queen" have similar vectors, with relationships between words reflected in the vector arithmetic (e.g., vking−vman+vwoman≈vqueen)

Visualization of embeddings in a 3D space shows clusters of semantically related words. The vectors are often reduced to three dimensions using techniques like PCA (Principal Component Analysis) or t-SNE (t-Distributed Stochastic Neighbor Embedding).

This mathematical framework allows word embeddings to underpin numerous NLP applications such as sentiment analysis, machine translation, and text classification, significantly improving the understanding and processing of human language.

Following is a visualization of an example of embedding using word2Vec.



References

[1] Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

# [2] Computer Vision: Algorithms and Applications, 2nd ed. 2022 [Richard Szeliski](https://szeliski.org/), The University of Washington.