

Optimization Algorithms in Machine Learning: A Comprehensive Guide to Understand the concept and Implementation.

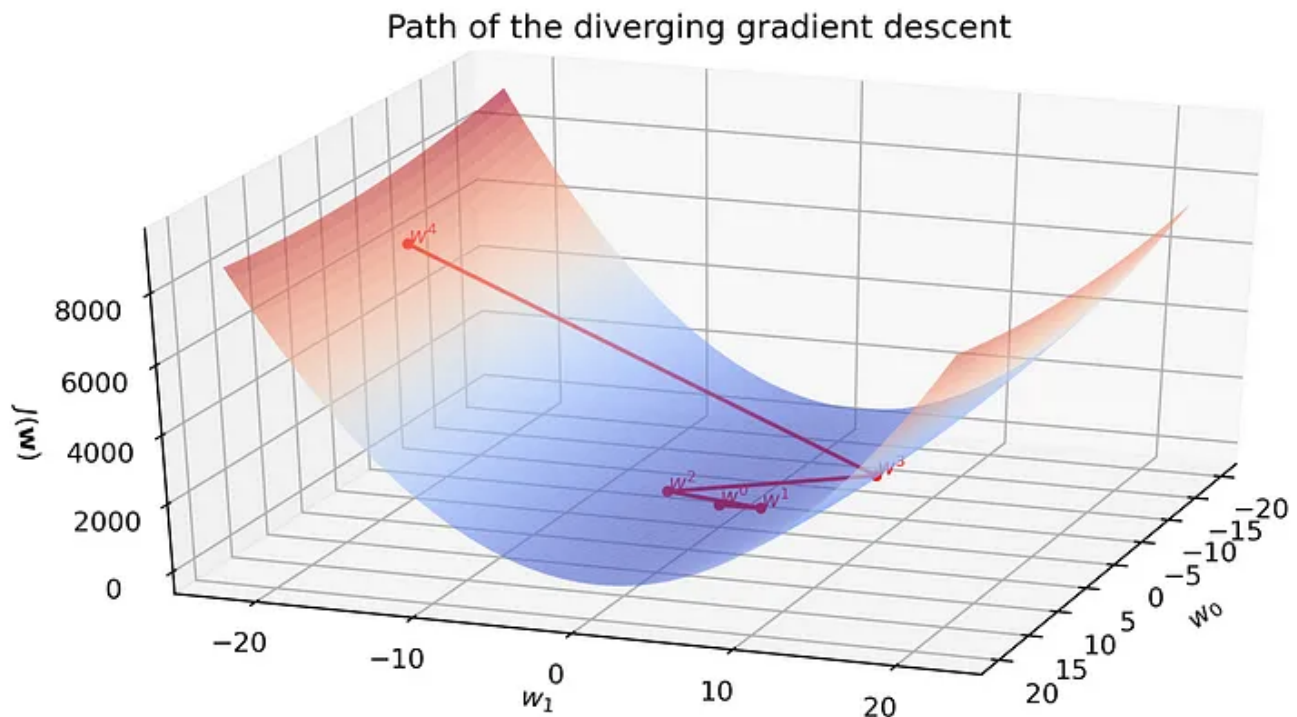


Koushik · [Follow](#)

9 min read · Dec 6, 2023



Optimizer, the engine of machine learning.

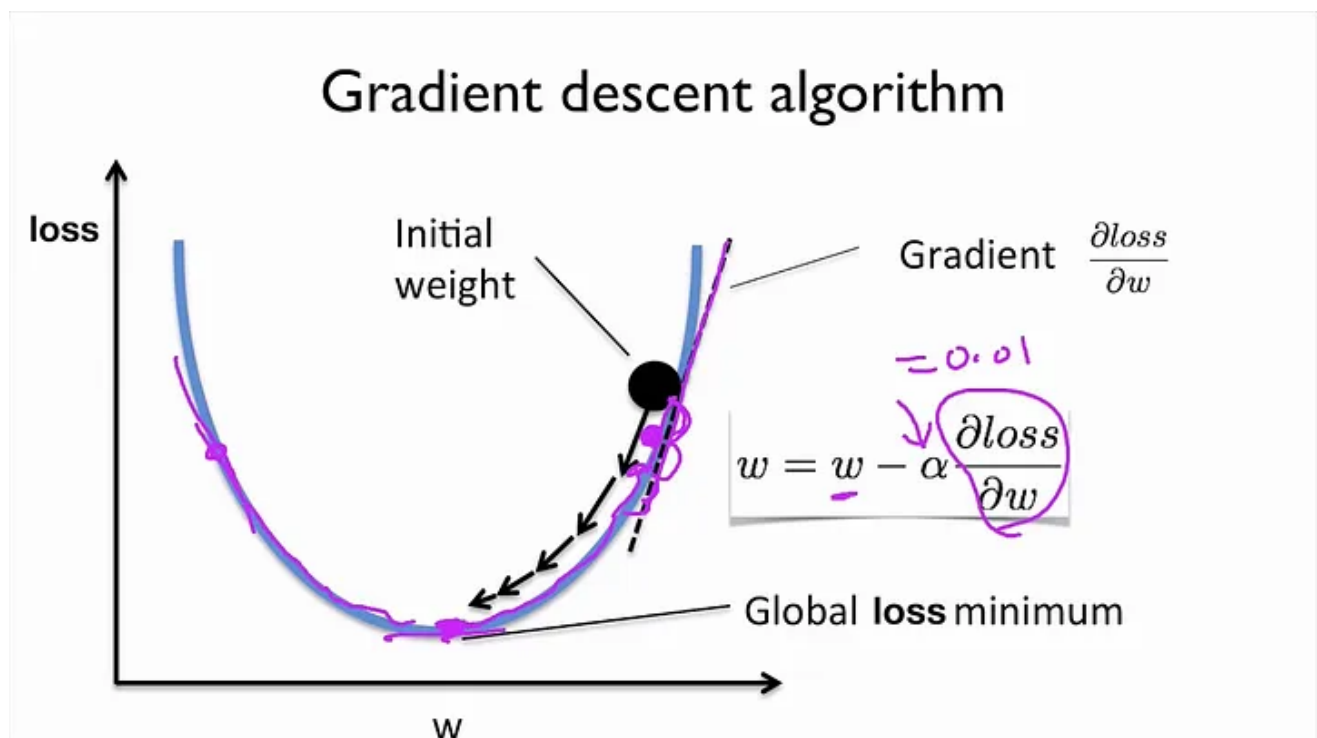


gradient descent. source:dmitrijskass

Definition: In the context of machine learning, optimization refers to the process of adjusting the parameters of a model to minimize (or maximize) some objective function. The objective function is often a measure of how well the model performs on a task, such as minimizing the error on a set of training data. The process involves finding the optimal set of parameters that result in the best performance of the model.

There are various optimization algorithms used in machine learning to find the optimal set of parameters. These algorithms are responsible for updating the model parameters iteratively during the training process. Some common optimization algorithms include:

Gradient Descent: Gradient Descent is a first-order iterative optimization algorithm widely used in machine learning and optimization problems. Its primary purpose is to minimize a differentiable cost or loss function by iteratively adjusting the parameters of a model.



$$w = w - \alpha * \delta loss / \delta w$$

Start with an initial set of parameter values (weights) for the model. In each iteration, compute the gradient of the loss function with respect to the parameters. where w represents the parameters, α is the learning rate, and $\delta loss / \delta w$ is the gradient of the objective function. Simplify the function you would like to minimize by using the first-order polynomial. This expression $\delta loss / \delta w$ represents the rate of change of the loss with respect to the weight parameter and is a crucial component in optimization algorithms like gradient descent. The goal is to adjust the weights in the direction that minimizes the loss, and the derivative provides information about the slope of the loss function at a specific point in the parameter space.

Objective: Gradient Descent aims to find the minimum of a cost or loss function. This function represents the difference between the predicted values of the model and the actual values (labels) in a training dataset.

Convergence and Challenges: Convergence is achieved when the algorithm reaches a point where further iterations do not significantly change the parameters. However, challenges such as choosing an appropriate learning rate and dealing with **saddle points** and **local minima** can affect convergence.

Let's see how to code it.

```
import numpy as np

def gradient_descent(X, y, w, learning_rate, iterations):
    m = len(y)

    for _ in range(iterations):
        predictions = np.dot(X, w)
        errors = predictions - y
        gradient = np.dot(X.T, errors) / m
        #  $\alpha = \text{learning\_rate}$  and  $\text{gradient} = \partial \text{loss} / \partial w$ 
        w = w - learning_rate * gradient

    return w
```

. . .

Stochastic Gradient Descent (SGD):

Stochastic Gradient Descent (SGD) is an optimization algorithm used in machine learning and deep learning for training models. It is an extension of the standard gradient descent algorithm, where instead of computing the gradient of the entire dataset, SGD computes the gradient and updates the model parameters for each training example individually or in small batches.

Why Use SGD:

Computational Efficiency: Computing the gradient using the entire dataset can be computationally expensive, especially for large datasets. SGD allows for more frequent updates with lower computational cost.

Faster Convergence: Since updates are made more frequently, SGD often converges faster than traditional gradient descent, especially when dealing with non-convex loss functions.

Memory Efficiency: Working with individual or mini-batches of data requires less memory compared to the entire dataset, making SGD suitable for cases where memory is a constraint.

Escape Local Minima: The stochastic nature of SGD introduces randomness in the optimization process, helping the algorithm escape local minima and explore the parameter space more effectively.

It's an extension of gradient descent where the parameters are updated using a single randomly chosen data point at a time, making it computationally less expensive.

$$\theta_{new} = \theta_{old} - \alpha \nabla J_i(\theta_{old})$$

Stochastic gradient descent

where i represents the randomly chosen data point. Let's see how to code it.

```
import numpy as np

def stochastic_gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)

    for _ in range(iterations):
        for i in range(m):
            rand_index = np.random.randint(0, m)
            X_i = X[rand_index, :].reshape(1, -1)
            y_i = y[rand_index]
            prediction = np.dot(X_i, theta)
            error = prediction - y_i
            gradient = X_i.T * error
```

```
theta = theta - learning_rate * gradient

return theta
```

In summary, SGD is advantageous for its computational efficiency, faster convergence, and ability to handle large datasets. However, it introduces variance in updates, and the choice of an appropriate learning rate is crucial to balance exploration and exploitation during optimization. The decision to use traditional gradient descent or SGD depends on factors such as dataset size, computational resources, and the nature of the optimization problem.

• • •

RMSprop (Root Mean Square Propagation): RMSprop is an optimization algorithm designed to address some of the limitations of traditional gradient descent, especially when dealing with non-convex and poorly conditioned optimization problems. It is particularly useful for problems with sparse data and helps to adaptively adjust the learning rates for different parameters.

How RMSprop Works: RMSprop adapts the learning rates of each parameter individually by dividing the learning rate by the root mean square of recent gradients. This allows the algorithm to automatically decrease the learning rate for parameters that have steep and rapidly changing gradients and increase the learning rate for parameters with small or slowly changing gradients.

Algorithm gist:

For each iteration t , compute the gradient ∇J_t of the loss function with respect to the parameters.

Update the exponentially decaying average of squared gradients:

$$v_t = \beta \cdot v_{t-1} + (1 - \beta) \cdot (\nabla J_t)^2$$

Update the parameters using the root mean square of the recent gradients:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t} + \epsilon} \cdot \nabla J_t$$

```
def RmsProp(X, y, theta, learning_rate, beta=.9, iterations, epsilon=1e-7):

    v_t = np.zeros_like(theta)
    # RMSprop optimization loop
    for _ in range(iterations):
        # Compute the gradient of the loss function
        predictions = X.dot(theta)
        errors = predictions - y
        gradient = X.T.dot(errors) / len(y)

        # Update the exponentially decaying average of squared gradients
        v_t = beta * v_t + (1 - beta) * gradient**2

        # Update the parameters using RMSprop
        theta -= (learning_rate / (np.sqrt(v_t) + epsilon)) * gradient

    return theta
```

In summary, RMSprop helps mitigate issues like slow convergence and oscillations in the learning rate by adapting the learning rates for each parameter based on the historical gradients. It is a popular choice in practice, especially for training deep neural networks, and can be particularly effective in scenarios where the landscape of the optimization problem is challenging or has varying curvature. The algorithm provides a balance between the aggressive adaptability of methods like AdaGrad and the more stable behavior of methods like SGD.

Adam (Adaptive Moment Estimation): Adam is an optimization algorithm used for training machine learning models. It combines ideas from both momentum optimization and RMSprop (Root Mean Square Propagation). Adam adapts the learning rates for each parameter individually based on the historical gradients of those parameters.

Why Adam ??

Adaptive Learning Rates: One of the key advantages of Adam is its adaptive learning rate mechanism. It adjusts the learning rates for each parameter based on the historical gradients, allowing it to perform well across different types of parameters and features.

Efficiency in Sparse Gradients: Adam is well-suited for sparse gradients, which is common in tasks like natural language processing. It maintains a separate adaptive learning rate for each parameter, making it less sensitive to the scale of the gradients.

Combining Momentum and RMSprop: Adam combines the momentum term to accelerate convergence in the parameter space with the RMSprop term to adaptively scale the learning rates. This combination helps handle noisy gradients and navigate through saddle points efficiently.

Effective in a Wide Range of Applications: Adam has shown effectiveness in a wide range of deep learning applications and is widely used in practice due to its robust performance and ease of use.

Algorithm

Algorithm 1: The Basic Adam Optimizer

Require:

- 1: Initialized parameter θ_0 , step size η , batch size N_B
- 2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^N$

Initialize: $m_0 = 0, v_0 = 0$

3: **For** all $t = 1, \dots, T$ **do**

4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{N_B}$ from dataset

5: $g_t \leftarrow \sum_{k=1}^N \nabla \{ (x_{ik}, y_{ik}, \theta_{t-1}) \} // f'(\theta_{t-1})$

6: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t // \text{moving Average}$

7: $v_t \leftarrow \beta_2 \cdot m_{t-1} + (1 - \beta_2) \cdot g_t^2$

8: $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}, \hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t} // \text{correction bias}$

9: $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$

10: **end for**

11: **return** final parameter θ_T

Adam algorithm. source: [springer](#)

Explanation of Adam parameters:

First Moment Estimate (m_t): This moving average is similar to the momentum term and represents the exponentially decaying average of past gradients. It helps the optimizer continue moving in the right direction even if the gradient updates become noisy.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla J_t$$

Second Moment Estimate (v_t): This moving average is similar to the squared gradients used in RMSprop. It represents the exponentially decaying average of the past squared gradients and helps adapt the learning rates for each parameter.

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla J_t)^2$$

The updated parameter (θ_{t+1}) is then calculated using the following formula:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t} + \epsilon} \cdot m_t$$

- α is the learning rate.
- ϵ is a small constant to prevent division by zero.
- t is the iteration step.
- J_t is the gradient of the loss function with respect to the parameters at iteration t .

Let's see how to code:

```
import numpy as np

def adam(X, y, theta, learning_rate, iterations, beta1=0.9, beta2=0.999, epsi
```



```

m = len(y)
m_t = np.zeros(theta.shape)
v_t = np.zeros(theta.shape)
t = 0

for _ in range(iterations):
    for i in range(m):
        t += 1
        X_i = X[i, :].reshape(1, -1)
        y_i = y[i]
        prediction = np.dot(X_i, theta)
        error = prediction - y_i
        gradient = X_i.T * error

        m_t = beta1 * m_t + (1 - beta1) * gradient
        v_t = beta2 * v_t + (1 - beta2) * (gradient**2)

        m_t_hat = m_t / (1 - beta1**t)
        v_t_hat = v_t / (1 - beta2**t)

        theta = theta - learning_rate * m_t_hat / (np.sqrt(v_t_hat) + eps)

    return theta

```

In summary, Adam combines the advantages of momentum and RMSprop to provide an adaptive and efficient optimization algorithm. It has become a popular choice for training deep neural networks due to its robust performance and ease of use.

Run this **main function** to compare the synthetic data to several optimizers. Furthermore, remember to tune hyperparameters and use real-world datasets.

```

import numpy as np

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred)**2)

# Generate synthetic data
np.random.seed(40)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Add bias term to X
X_b = np.c_[np.ones((100, 1)), X]

```

```

# Initial parameters
theta_initial = np.random.randn(2, 1)

# Set hyperparameters
learning_rate = 0.003
iterations = 1000
batch_size = 32

# Apply optimization algorithms
theta_gd = gradient_descent(X_b, y, theta_initial, learning_rate, iterations)
theta_sgd = stochastic_gradient_descent(X_b, y, theta_initial, learning_rate,
theta_rmsprop = RmsProp(X_b, y, theta_initial, learning_rate, beta=.9, iterat
theta_adam = adam(X_b, y, theta_initial, learning_rate, iterations)

# Evaluate the models
predictions_gd = np.dot(X_b, theta_gd)
predictions_sgd = np.dot(X_b, theta_sgd)
predictions_rmsprop = np.dot(X_b, theta_rmsprop)
predictions_adam = np.dot(X_b, theta_adam)

# Print Mean Squared Error for each model
print("Mean Squared Error (Gradient Descent):", mean_squared_error(y, predict
print("Mean Squared Error (Stochastic Gradient Descent):", mean_squared_error
print("Mean Squared Error (RmsProp):", mean_squared_error(y, predictions_rmsp
print("Mean Squared Error (Adam):", mean_squared_error(y, predictions_adam))

```

Thank You!

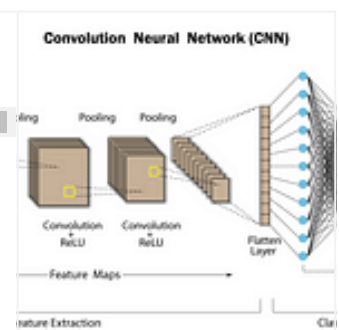
. . .

Feel free to visit my other articles..

Understanding Convolutional Neural Networks (CNNs) in Depth

Convolutional Neural Networks skillfully capturing and extracting patterns from data, revealing the hidden artistry...

medium.com



Unfolding the Intelligence: Machine Learning vs. Deep Learning

Introduction:

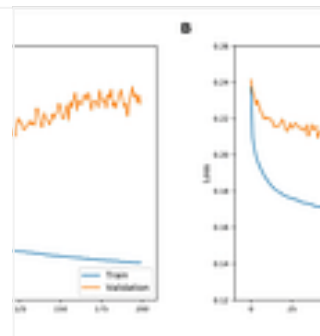
medium.com



What is Overfitting and Underfitting , and how to deal with it step by step?

In machine learning, it is common to face a situation when the accuracy of models on the validation data would peak...

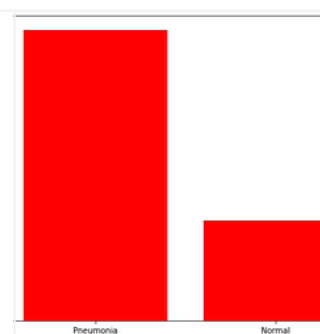
medium.com



What Class Imbalance is all about: details with examples

Class Imbalance is a common problem in machine learning, especially in classification problems. Imbalance data can...

medium.com



• • •

References

Understanding Activations & Optimization- Neural Networks

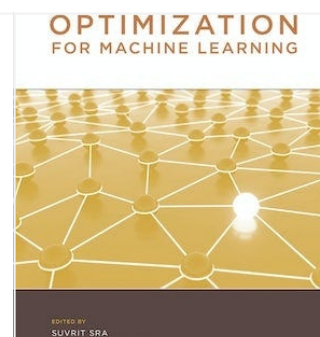
Edit description

community.ibm.com

Optimization for Machine Learning

An up-to-date account of the interplay between optimization and machine learning, accessible to students and...

mitpress.mit.edu



12.1. Optimization and Deep Learning - Dive into Deep Learning 1.0.3 documentation

In this section, we will discuss the relationship between optimization and deep learning as well as the challenges of...

d2l.ai

Optimization

Machine Learning

Deep Learning

Data Science

Optimization Algorithms

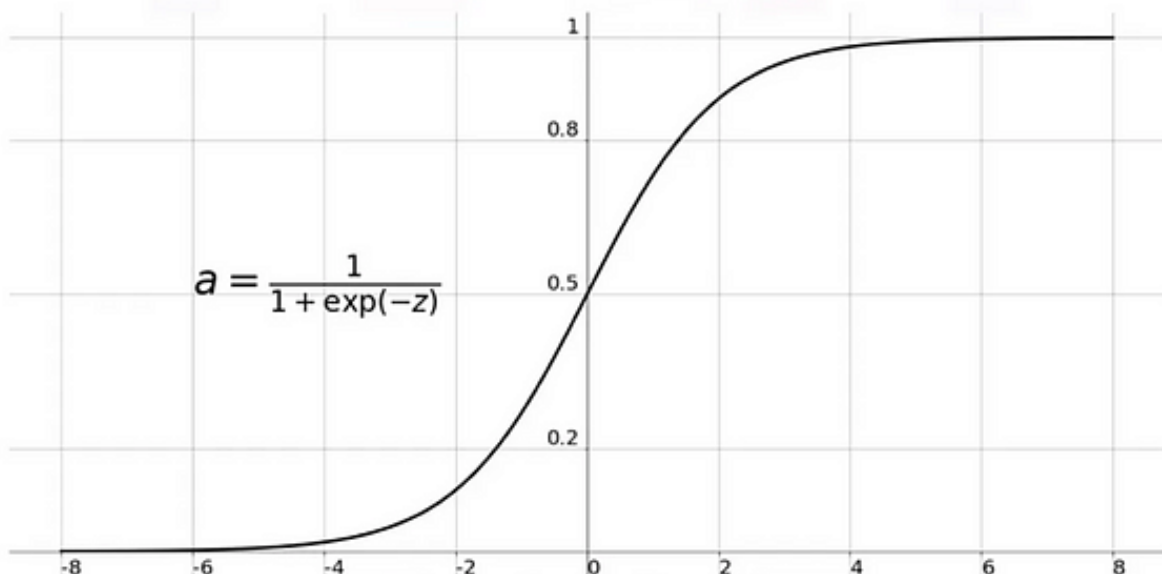


Written by Koushik

220 Followers

Machine Learning Engineer

More from Koushik

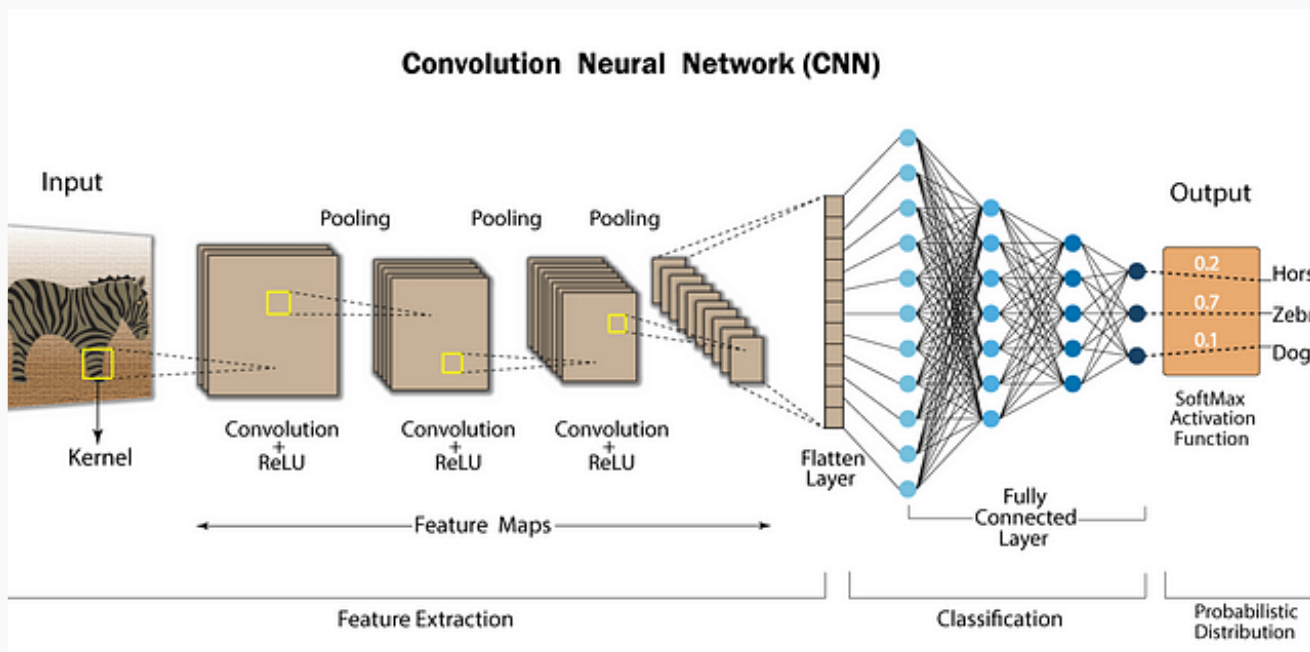



Koushik

Logistic Regression From Scratch

Logistic regression is often mentioned in connection to classification tasks. The model is simple and one of the easy starters to learn...

Aug 28, 2023 🖱️ 91 💬 1

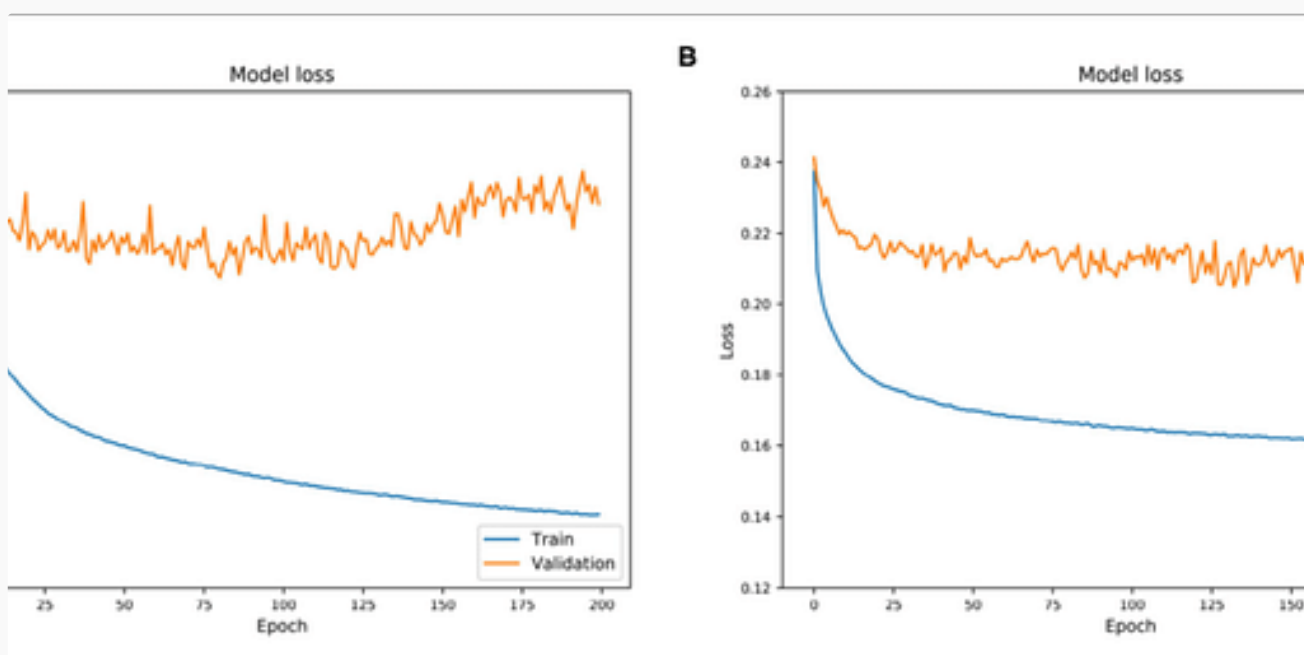


 Koushik

Understanding Convolutional Neural Networks (CNNs) in Depth

Convolutional Neural Networks skillfully capturing and extracting patterns from data, revealing the hidden artistry within pixels.

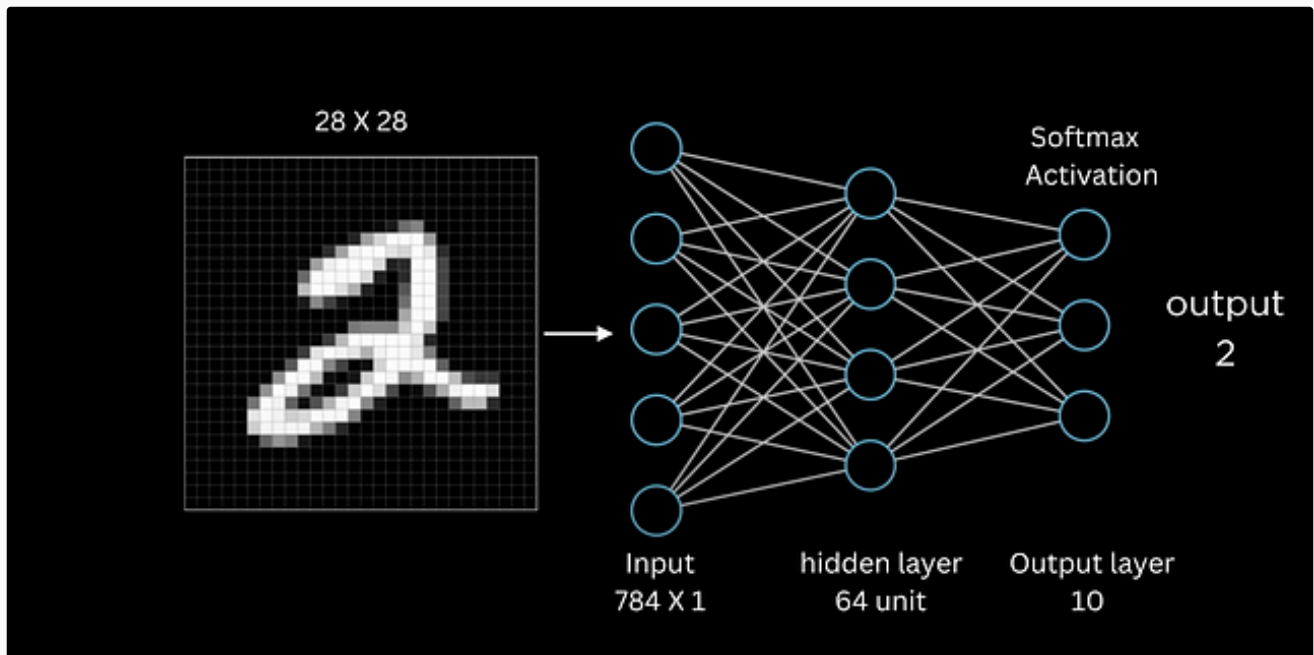
Nov 28, 2023 🖱️ 616 💬 8



What is Overfitting and Underfitting , and how to deal with it step by step?

In machine learning, it is common to face a situation when the accuracy of models on the validation data would peak after training for a...

Sep 20, 2023 🖱 108 💬 1



MNIST(hand written digit) Classification Using Neural Network From Scratch

What is Neural Network ?

Aug 31, 2023 🖱 67 💬 1



See all from Koushik

Recommended from Medium

AMAZON.COM

Software Development Engineer

Seattle, WA

Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by \$25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay



Alexander Nguyen in Level Up Coding

The resume that got a software engineer a \$300,000 job at Google.

1-page. Well-formatted.



Jun 1

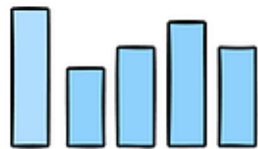


20K

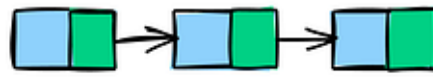


378



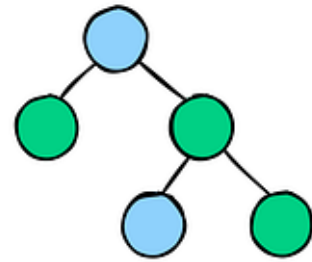


SORTING

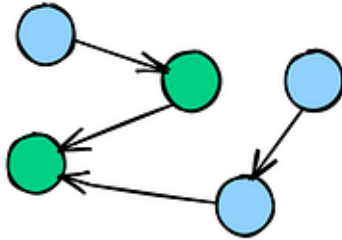


LINKED LIST

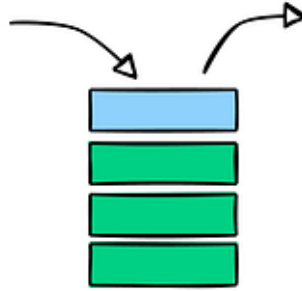
blog.algomaster.io



TREE



GRAPH



STACK



QUEUE



Ashish Pratap Singh in AlgoMaster.io

How I Mastered Data Structures and Algorithms

Getting good at Data Structures and Algorithms (DSA) helped me clear interviews at Amazon, Google and Microsoft.



Jul 23



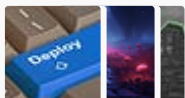
1.2K



13



Lists



Predictive Modeling w/ Python

20 stories · 1503 saves



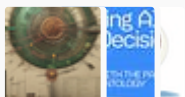
Practical Guides to Machine Learning

10 stories · 1832 saves



Natural Language Processing

1687 stories · 1258 saves



data science and AI

40 stories · 237 saves



Vyacheslav Efimov in Towards Data Science

Understanding Deep Learning Optimizers: Momentum, AdaGrad, RMSProp & Adam

Gain intuition behind acceleration training techniques in neural networks

Dec 30, 2023 🖱 434 💬 5

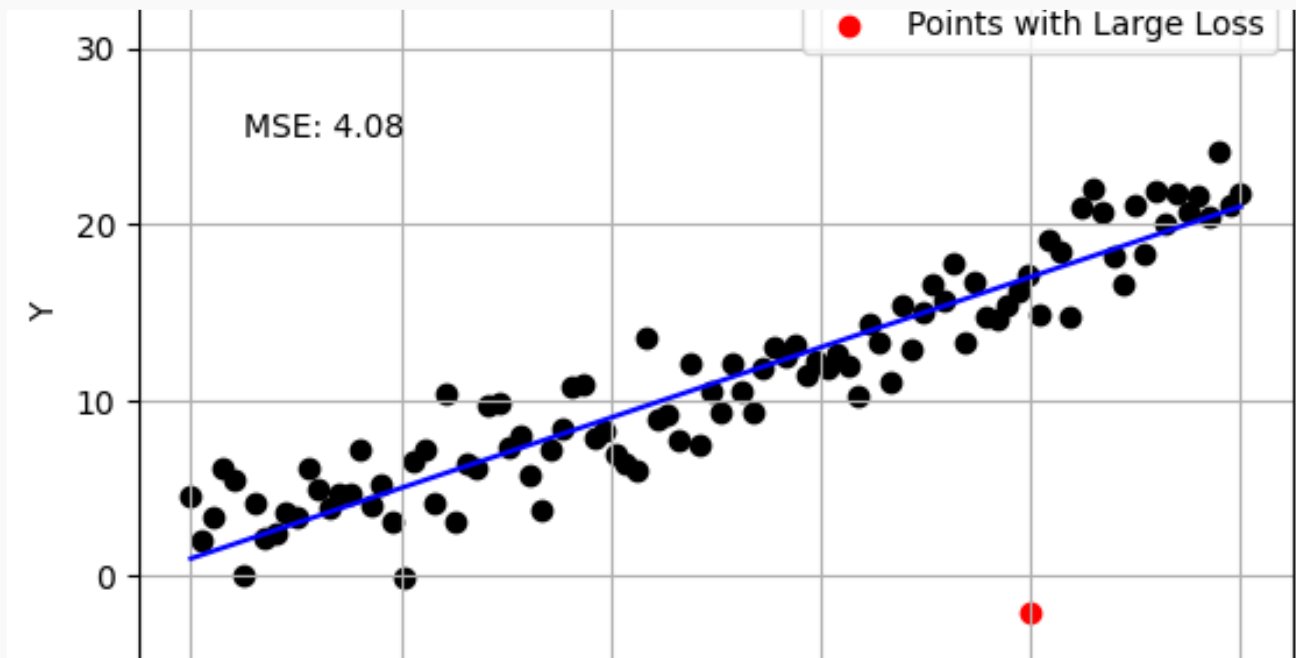


AI SageScribe

Understanding Causal Self-Attention: The Key to Autoregressive Models

Causal self-attention, also known as masked self-attention, is a fundamental concept in transformer models, particularly in autoregressive...

★ Aug 8

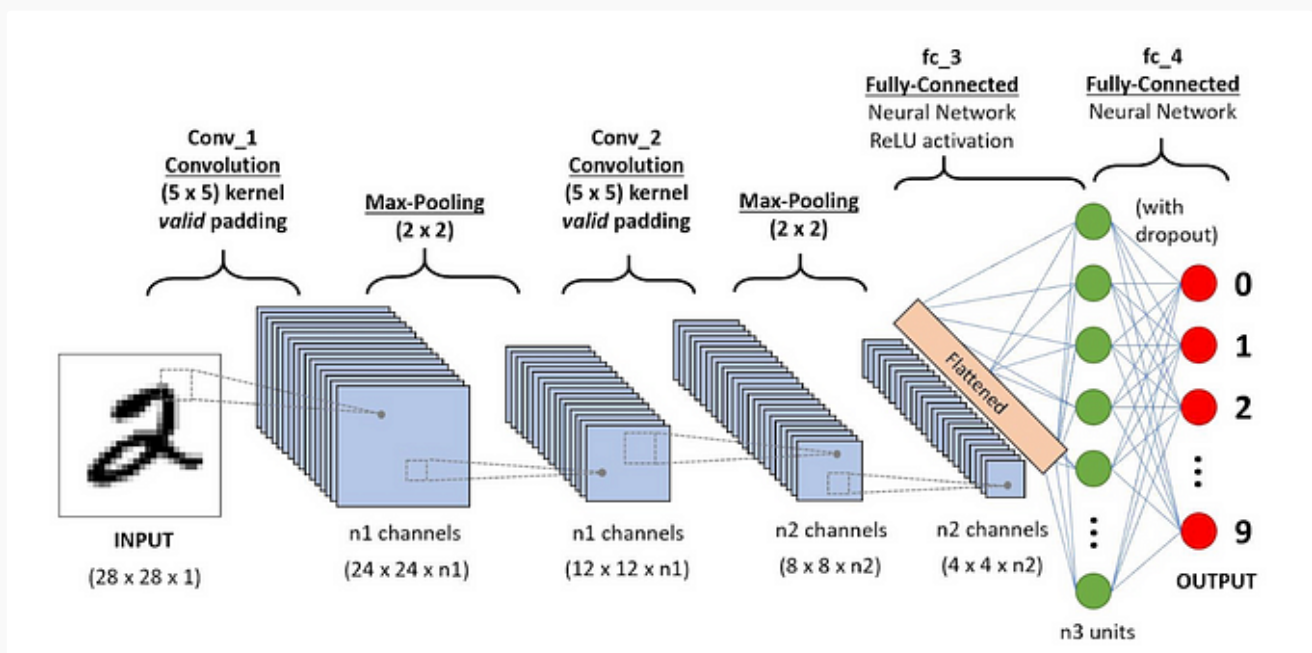


👤 Swayam

Choosing the Right Regression Loss Function: A Guide

Introduction

Apr 8 🖱️ 6





Luqman Zaceria in Lumos

Understanding Convolutional Neural Networks (CNNs)

Hey everyone! We're going to explore one of the most influential and powerful tools in the world of deep learning: Convolutional Neural...



Aug 5



71



See more recommendations