

Fundamental principles of Machine Learning: A technical deep dive using linear regression



Table of contents

Abbreviations	3
Introduction	4
Problem statement	4
Business challenges and industry needs	5
Related approaches	6
Solution	6
Problem formulation and data representation	7
Linear regression as a supervised learning model	9
Loss function: Mean Squared Error (MSE)	10
Optimization via Batch Gradient Descent (BGD)	10
Training results and model evaluation	14
Benefits	15
Discussion	16
Conclusion	16
References	17
Author information	17

Abbreviations

LLM	Large Language Model
BGD	Batch Gradient Descent
SGD	Stochastic Gradient Descent
MSE	Mean Square Error
RMSE	Root Mean Square Error
R^2	R-squared or Coefficient of Determination

Introduction

Machine Learning (ML) systems are often introduced through the lens of highly complex architectures such as deep neural networks, transformer-based Large Language Models (LLMs) and reinforcement learning agents. While these models define the frontier of modern AI applications, the fundamental principles governing their behavior – forward pass, backward pass, loss minimization and model evaluation – are most effectively understood through simpler models. At its core, machine learning is not defined by architectural complexity, but by mathematical structure and optimization principles. Among foundational models, linear regression stands as one of the most instructive and enduring frameworks in supervised learning theory.

Problem statement

The information technology industry is rapidly accelerating toward large-scale adoption of AI and ML-based systems to improve operational efficiency, optimize costs, enable innovations that are infeasible without AI and create new revenue streams. Enterprises of all sizes are increasingly deploying systems powered by deep neural networks, transformer-based LLMs and reinforcement learning agents. Despite the rapid proliferation of ML systems across industries, there exists a growing disconnect between the widespread use of advanced models and the depth of understanding required to design, debug and maintain them effectively. While complex models such as deep neural networks, transformer-based large language models and reinforcement learning agents deliver strong empirical performance, their architectural and optimization complexity often masks the foundational learning mechanisms underlying their behavior.

Moreover, in many organizations, ML development has increasingly become framework-driven rather than principle-driven. Engineers can train and deploy models using high-level libraries and automated pipelines without fully engaging with foundational constructs such as loss formulation, gradient-based optimization, parameter update rules, learning rate dynamics and evaluation metrics. As a result, ML systems are frequently treated as black boxes, limiting practitioners' ability to reason systematically about model behavior when performance deviates from expectations.

This creates a clear need for a structured, technically grounded resource that revisits the fundamental principles of machine learning through a simple yet complete model. Addressing this gap is critical to building transferable understanding, strengthening debugging and explainability capabilities, and enabling practitioners to reason confidently across both foundational and advanced learning architectures.

Business challenges and industry needs

Despite the widespread AI adoption, many organizations continue to face structural challenges in harnessing the full potential of AI and ML at scale. A primary limiting factor is the shortage of talent with a deep and transferable understanding of foundational ML principles.

Several recurring challenges emerge when core principles governing ML algorithms are not sufficiently understood:

- **Talent development and knowledge transfer challenges** As demand for ML adoption and system development accelerates, organisations struggle to onboard, upskill and scale engineering teams with the required depth of expertise. Introducing advanced architectures without first establishing a strong grounding in foundational principles often results in conceptual gaps, fragmented understanding and inconsistent engineering practices across teams. A clear, technically grounded exploration of ML fundamentals – anchored in linear regression – enables more effective training, strengthens conceptual clarity and supports sustainable talent development at scale.
- **Difficulty debugging and explaining model behavior** In production environments, ML systems frequently encounter issues such as unexpected prediction errors, performance degradation on new data and instability during optimization. Without a clear understanding of fundamental ML concepts – including loss functions, parameter optimization techniques and evaluation metrics – debugging of ML systems becomes trial-and-error. Businesses require a conceptual framework that explains why models behave the way they do. Linear regression offers a controlled and transparent setting in which these core mechanisms can be systematically analyzed, interpreted and understood before being extended to more complex architectures.
- **Gaps between academic theory and practical implementation** Although theoretical ML concepts are well-documented, practitioners often find it challenging to connect mathematical formulations with implementation details such as data preparation, forward pass, backward pass, loss minimization, gradient computation, model parameter update rule and evaluation metrics. There is a clear industry need for technically rigorous resources that bridge theory and practice through models that are simple in structure yet complete in formulation. Such models enable practitioners to internalize principles that scale directly to real-world enterprise-grade systems.

By revisiting foundational ML concepts through the lens of linear regression, this whitepaper seeks to address the knowledge gap by guiding the reader through model formulation, loss minimization, parameter optimization, model training and evaluation techniques.

Related approaches

The foundational principles of ML have been explored extensively across academic literature, educational resources and industry-oriented materials. These approaches can be broadly categorized into academic and theoretical treatments, framework-driven tutorials and deep learning-centric based pedagogical approaches. While each of these approaches contributes valuable insight, they often fall short of delivering a unified, principle-driven understanding that connects mathematical foundations with practical system behavior.

- **Academic and theoretical treatments**

Classical ML textbooks and academic courses frequently introduce linear regression as an entry point to supervised learning. These treatments are mathematically rigorous and provide strong theoretical grounding. However, they often abstract away the operational aspects of model training, including iterative optimization, learning rate dynamics and empirical evaluation. As a result, practitioners may develop a sound understanding of the mathematics but not how those formulations translate into practical model training, debugging and performance tuning in real-world environments.

- **Framework-driven tutorials and industry documentation**

Industry documentation and online tutorials typically emphasizes rapid model development using high-level ML frameworks such as Scikit-learn, Keras, PyTorch. While highly effective for accelerating experimentation and adoption, this approach frequently abstracts critical concepts including loss minimization, gradient computation and parameter update rules. Practitioners trained predominantly through framework-driven materials may therefore struggle to explain model behavior, diagnose training failures or transfer knowledge across different architectures and learning paradigms. The emphasis on tooling can unintentionally overshadow the principles that govern learning dynamics.

- **Deep learning based pedagogical approaches**

Many modern ML educational resources adopt a deep-learning-first approach, introducing multilayer neural networks, backpropagation and optimization early in the learning process. While this reflects current industry practice, it increases learning complexity and can overwhelm new practitioners who lack a firm grasp of foundational ML principles typically better acquired through simpler and canonical models.

Solution

This whitepaper presents linear regression as a fundamental working ML system that exposes the complete learning pipeline in an explicit and interpretable manner. Rather than treating it as a simplistic technique, we use it as a fully structured model through which every stage of the ML lifecycle can be examined with clarity and technical precision.

The approach systematically progresses through each phase of model

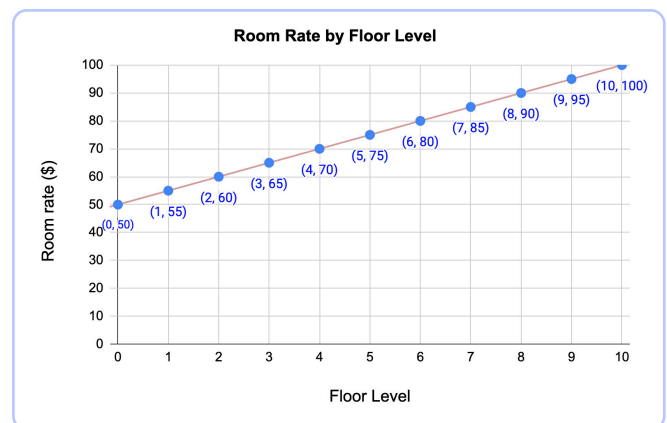
development – beginning with problem definition, data representation and model formulation. It then advances to loss function specification using Mean Squared Error (MSE), parameter optimization via Batch Gradient Descent (BGD), and concludes with structured model evaluation using MSE, Root Mean Squared Error (RMSE) and the Coefficient of Determination (R^2).

By articulating each component of the learning process end-to-end, this framework enables practitioners to internalize the mechanics of model behavior in a way that directly generalizes to more complex ML architectures.

Problem formulation and data representation

Before diving into ML concepts, let's look at the data of an imaginary hotel, "Hotel Bay View," which has all rooms facing a scenic bay. The hotel has priced its room rates based on the floor level. The room rates increase with the increase in floor level as the view gets better at higher levels. Please note that later we will use this hotel data for deep diving into ML principles using a linear regression model. In the real world, hotel room pricing is not as simple as this, but we are keeping it as simple as possible to use a linear regression model for exploring the fundamentals of ML.

Floor Level	Room rate (\$)
0	50
1	55
2	60
3	65
4	70
5	75
6	80
7	85
8	90
9	95
10	100



Plotting the room rate and floor level on a graph, we observe that there is a linear relationship between the room rate and floor level. Based on the above graph, clearly, room rate and floor levels have a linear relationship. In terms of algebra, that relationship can be represented using the equation of a line.

The relationship between room rate and floor levels can be represented as:

$$\text{Room_rate} = m * \text{Floor_level} + b$$

Shortly, we will look at what m and b are in this relationship. For brevity, let's represent the room rate with y and the floor level with x .

The room rate and floor level linear relationship can now be rewritten as $y = m*x + b$

- m is the rate of change of the room rate with respect to the change in floor levels. m is also called the slope of the line. It can be calculated

as $m = (y_2 - y_1) / (x_2 - x_1)$ if y_2 is the room rate for floor level x_2 and y_1 is the room rate for floor level x_1 .

- b is a constant called the intercept or bias. It is the value of y when $x = 0$. Consider it as the base room rate when the floor level is 0.

Calculating slope m :

- $m = (60 - 55) / (2 - 1)$ using the room rates of floor level 1 and 2.
- $m = 5$

Finding value of b : Intercept b is the value of y when $x = 0$. In the above graph, we can see that when $x = 0$ then $y = 50$.

- $y = m * x + b$
- $50 = 5 * 0 + b$
- $b = 50$

Now that we know the value of m and b the

- Room_rate = $m * \text{Floor_level} + b$ becomes
- Room_rate = $5 * \text{floor_level} + 50$
- $y = 5 * x + 50$

Using linear algebra and a graph, we found out the mathematical relationship between the floor level (independent variable) and the room rate (dependent variable). We can now utilize this relationship to find room rate of any new floor level, say floor level 63, simply by doing $y = 5 * 63 + 50 = 365$.

x (Floor Level)	y = 5*x + 50 (Room Rate in \$)
0	50
1	55
2	60
3	65
4	70
5	75
6	80
7	85
8	90
9	95
10	100

However, we want to use ML techniques to find out the relationship between the room rate and floor level from the given data of room rates and floor levels and predict the room rate for any new floor level. In this problem, the input is the floor level (x) and the output (y) is the room rate. We need to predict a continuous value (room rate) from the input (floor level) and not classify the input into some categories. Hence, this is a regression problem, not a classification problem. In this problem, we have data in which the output (room rate) is known for an input (floor level).

The data that has known input-output pairs is called **labeled data** in ML lingo. When labeled data is used to train an ML model, it

is called **supervised ML**. In terms of ML problem framing, our problem can be framed as a **supervised linear regression problem to predict room rate based on floor level**.

In this problem, in the non-ML world, the input is floor level (x), and the output (y) is the room rate. In ML lingo, the input (x) is also called a **feature** and the known output, or the desired output (y) is called the **target**. The ML model's predicted result is called output. Let's represent the ML model's

predicted output as \hat{y} , pronounced “y hat”. We will see later how the ML model predicts the output \hat{y} .

For supervised ML, data is essential for model training and evaluation. We have the hotel data on the floor level (independent variable) and room rate (dependent variable) for a total of 11 floors. We will split the full dataset into at least two sets: the training dataset and the test dataset. The **training dataset** will be solely used for training and the **test dataset** will be used for evaluating the performance of the ML model.

Full dataset

Floor level	Room rate (\$)
Input	Desired output
Feature	Target
x	y
0	50
1	55
2	60
3	65
4	70
5	75
6	80
7	85
8	90
9	95
10	100

Training dataset

Floor level	Room rate (\$)
Input output	Desired Feature
Target	
x	y
0	50
1	55
2	60
3	65
4	70
5	75
6	80

Test dataset

Floor level	Room rate (\$)
Input	Desired output
Feature	Target
x	y
7	85
8	90
9	95
10	100

Linear regression as a supervised learning model

In supervised ML, the mathematical relationship between the input (feature) and the predicted output is the **ML model**. It is represented as follows for a linear regression model having only one variable input (x).

- $\hat{y} = w * x + b$ Where \hat{y} is the ML model predicted output (predicted room rate) w is called weight and is the ML world equivalent of slope m b is bias or intercept. It is the value of \hat{y} when $x = 0$ The weight (w) and bias (b) are called **model parameters**

At this stage we know our ML model is $\hat{y} = w * x + b$ for predicting the room rate based on the floor level (x), but we don't yet know the value of weight (w) and bias (b). We need to train the ML model to find the weight and the bias. In ML lingo, the weight (w) and bias (b) are called model parameters. The supervised **ML model training** is the process of learning the optimal weight and bias that minimizes the **prediction error**. The prediction error is the difference between the predicted output (\hat{y}) and the desired output or the target (y). The lower, the prediction error, the higher the accuracy of the prediction.

- Prediction error = $y - \hat{y}$

In our training data set, there are 7 data points. We will use this training dataset for training our model in section BGD.

Floor level	Room rate (\$)
Input	Desired output
Feature	Target
X	Y
0	50
1	55
2	60
3	65
4	70
5	75
6	80

Loss function: MSE

A **loss function or cost function** is utilized to measure how wrong or erroneous the model prediction (\hat{y}) is compared to the desired output or the target (y). For linear regression, **MSE** is commonly used as a loss function. The goal of training our ML model ($\hat{y} = w * x + b$) is to find the optimal value of parameters weight (w) and bias (b) that minimizes the MSE loss function.

- MSE loss function, $L_MSE(w,b)$
 - $L_MSE = (1/N) * \sum (y_i - \hat{y}_i)^2$ where $\hat{y}_i = wx_i + b$
 - $L_MSE = (1/N) * \sum (y_i - (wx_i + b))^2$ By substituting \hat{y}_i with $wx_i + b$.

Where ,

- N is the total number of data points in the dataset.
- y_i is the desired output for the i-th data point.
- \hat{y}_i is the predicted output for the i-th data point.
- \sum is Summation of all the squared prediction errors for $i=1$ to N
- $(y_i - \hat{y}_i)$ is the prediction error for a single data point i
- $(y_i - \hat{y}_i)^2$ is the squared error for a single data point i
- $\sum (y_i - \hat{y}_i)^2$ is the Sum of Squared Errors (SSE) over all N data points.
- $(1/N) * \sum_i (y_i - \hat{y}_i)^2$ is the average of the squared errors over all N data points. This entire expression is the MSE.

Optimization via Batch Gradient Descent (BGD)

To minimize the MSE loss function, there are **optimization algorithms** such as Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD), Adam, etc. For our linear regression model training, we will use the BGD optimization algorithm to find the optimal value of parameters weight (w) and bias (b) that minimizes the MSE loss function $L_MSE(w,b) = (1/N) * \sum (y_i - (wx_i + b))^2$. And, once we have found the optimal value of parameters weight (w) and bias (b), we can say that the ML model ($\hat{y} = w * x + b$) has been **trained**.

Let's walk through a step-by-step explanation of how the **BGD algorithm** works. We have essentially 5 steps. Steps 2 to 4 are repeated for a fixed number of iterations called epochs. In ML, each **epoch** is one complete pass of the entire training data set through the learning algorithm.

Step 1: Initialization

Initialize the parameters weight (w) and bias (b) with random initial guesses. Also, initialize the learning rate (α) described in Step 4 below.

- $w = 3$
- $b = 20$
- $\alpha = 0.01$

Step 2: Forward pass. Make predictions.

One by one, make the room rate prediction for the entire training data set. In our training data set, $N = 7$

The model prediction \hat{y}_i is done as

- $\hat{y}_i = wx_i + b$ for $i=1$ to N

In our problem, the predicted room rate $\hat{y}_i = wx_i + b$ for the i -th floor level.

	Floor level	Room rate (\$)	Weight	Bias	Prediction (\hat{y}) at each data point
	Feature	Target			
Index i	x	y	w	b	$\hat{y}_i = wx_i + b$
1	0	50	3	20	20
2	1	55	3	20	23
3	2	60	3	20	26
4	3	65	3	20	29
5	4	70	3	20	32
6	5	75	3	20	35
7	6	80	3	20	38

Step 3: Backward pass. Calculate the gradients.

The gradients are the partial derivatives of the MSE Loss function with respect to the parameters weight (w) and bias (b) for the entire training data set. The gradient of the MSE loss function is a vector that points in the direction of the greatest rate of increase of the function. The direction of the gradient will be very useful in the next Step 4.

In BGD algorithm using MSE loss function, the gradients of MSE loss function $L_MSE = (1/N) * \sum (y_i - (wx_i + b))^2$ with respect to weight w and bias b are -

- The gradient with respect to weight w is the partial derivative of L_MSE with respect to weight w .
 - $\partial L_MSE / \partial w = -(2/N) * \sum (x_i * (y_i - \hat{y}_i))$
 - The $\partial L_MSE / \partial w$ is the average of partial derivative of squared error $L_i = (y_i - \hat{y}_i)^2$ with respect to w over the entire training data set.

- The partial derivative of squared error $L_i = (y_i - \hat{y}_i)^2$ with respect to w for the i th data point is:
 $\partial L_i / \partial w_i = -(2) * (x_i * (y_i - \hat{y}_i))$
 - The $\partial L_MSE / \partial w$ can also be expressed as :
 $\partial L_MSE / \partial w = (1/N) * \sum \partial L_i / \partial w_i$ for $i=1$ to N
- The gradient with respect to bias b is the partial derivative of L_MSE with respect to bias b .
 - $\partial L_MSE / \partial b = -(2/N) * \sum (y_i - \hat{y}_i)$
 - The $\partial L_MSE / \partial b$ is the average of partial derivative of squared error $L_i = (y_i - \hat{y}_i)^2$ with respect to b over the entire training data set.
 - The partial derivative of squared error $L_i = (y_i - \hat{y}_i)^2$ with respect to b for i -th data point is:
 $\partial L_i / \partial b_i = -(2) * (y_i - \hat{y}_i)$
 - The $\partial L_MSE / \partial b$ can also be expressed as :
 $\partial L_MSE / \partial b = (1/N) * \sum \partial L_i / \partial b_i$ for $i=1$ to N
 - For our problem, we will first calculate the $\partial L_i / \partial b_i$ for each data point and then calculate the $\partial L_MSE / \partial b$.

Now using the training data set and predicted room rates (in Step 2), calculate the gradients of the MSE loss function with respect to weight w and bias b .

For epoch #1, here is an illustration of how the gradients ($\partial L_MSE / \partial w$ and $\partial L_MSE / \partial b$) of the MSE loss function with respect to weight w and bias b are calculated.

	Floor level	Room rate(\$)			From step 2	Step 3			
	Feature	Target	Weight	Bias	Room rate prediction (\hat{y}) at each data point	Gradient of loss L_i with respect to w at each data point	Gradient of L_MSE for w	Gradient of loss L_i with respect to b at each data point	Gradient of L_MSE for b
Index i	x	y	w	b	$\hat{y}_i = wx_i + b$	$\partial L_i / \partial w_i = -(2) * (x_i * (y_i - \hat{y}_i))$	$\partial L_MSE / \partial w = (1/N) * \sum \partial L_i / \partial w_i$	$\partial L_i / \partial b_i = -(2) * (y_i - \hat{y}_i)$	$\partial L_MSE / \partial b = (1/N) * \sum \partial L_i / \partial b_i$
1	0	50	3	20	20	0	-232	-60	-72
2	1	55	3	20	23	-64		-64	
3	2	60	3	20	26	-136		-68	
4	3	65	3	20	29	-216		-72	
5	4	70	3	20	32	-304		-76	
6	5	75	3	20	35	-400		-80	
7	6	80	3	20	38	-504		-84	

Hence, for epoch #1 the MSE loss function gradients are

- $\partial L_{MSE} / \partial w = -232$
- $\partial L_{MSE} / \partial b = -72$

Step 4: Update the parameters (w and b)

The gradient descent parameters update rule in ML is: new parameter = old parameter - (learning rate * gradient of parameter)

The **learning rate** is a scaling factor that controls the amount of change to be applied to the old parameter to get the new parameter value. Unlike parameters w and b , the learning rate (represented by α) is not learned by the ML model. The value of the learning rate (α) must be set before the ML training begins. In the ML lingo, the learning rate is a **hyperparameter**.

So, applying the above gradient descent parameters update rule to parameters w and b :

- new weight = old weight - learning rate * gradient of w
- new $w = \text{old } w - \alpha * (\partial L_{MSE} / \partial w)$
- new bias = old bias - learning rate * gradient of b
- new $b = \text{old } b - \alpha * (\partial L_{MSE} / \partial b)$

Note: The gradient $\partial L_{MSE} / \partial w$ and $\partial L_{MSE} / \partial b$ points in the direction of increase of the function; however, our objective is to minimize the MSE loss function. So, the gradient is subtracted to move in the direction of the decreasing loss function.

In epoch # 1, the values of old w and old b are the initial parameters weight (w) and bias (b) set in the Initialization Step 1.

$\text{new } w = \text{old } w - \alpha * (\partial L_{MSE} / \partial w)$	$\text{new } b = \text{old } b - \alpha * (\partial L_{MSE} / \partial b)$
$\text{new } w = 3 - 0.01 * (-232)$	$\text{new } b = 20 - 0.01 * (-72)$
$\text{new } w = 5.32$	$\text{new } b = 20.72$

So, in the first epoch, our new parameters weight (w) and bias (b) are 5.32 and 20.72 respectively.

Step 5: Repeat from Step 2 for a fixed number of iterations (e.g., 600 epoch).

The entire process from Step 2 to Step 4 is repeated for the next epoch with the new value of parameters weight (w) and bias (b) found in backward pass Step 3 until the final epoch. The value of parameters weight (w) and bias (b) at the final epoch are the learned values of model parameters optimized using the BGD algorithm. If the epoch is 600, then the final value of parameters weight w and bias b will be **5.1970 and 49.1653** respectively.

Training results and model evaluation

After the final training epoch (e.g., 600 epochs) in the above Step 5, we found that the learned values of the model parameters weight w and bias b , are **5.1970** and **49.1653**, respectively. Substituting these learned parameters into our linear regression model $\hat{y} = w * x + b$, the **trained model** can be expressed as :

$$\hat{y} = 5.1970 * x + 49.1653$$

The supervised linear regression model to predict room rate (\hat{y}) based on floor level (x) is now ready for predictions and evaluation.

The model evaluation is performed after training to assess how well the final learned parameters generalize to unseen data, rather than how well the model fits the training data used during optimization. The evaluation is typically carried out by computing prediction errors on the validation or **test dataset**.

Using our trained model and the held-out test dataset, the predicted room rates and the prediction errors can be calculated as follows.

Floor level	Room rate (\$)	Predicted room rate (\$)	Prediction error	Squared prediction error	Squared variance of target from mean
Input	Desired output	Predicted output			
Feature	Target				
x	y	$\hat{y} = 5.1970 * x + 49.1653$	$y - \hat{y}$	$(y - \hat{y})^2$	$(y - \bar{y})^2$
7	85	85.5443	-0.5443	0.2963	7
8	90	90.7413	-0.7413	0.5495	8
9	95	95.9383	-0.9383	0.8804	9
10	100	101.1353	-1.1353	1.2889	10

Common performance metrics used for model evaluation include MSE, Root Mean Squared Error (RMSE) and the Coefficient of Determination (R^2).

- **MSE** is calculated using the sum of squared prediction error. It measures the average squared difference between the predictions and the target values.
- **RMSE** is the square root of MSE. It measures the standard deviation of the prediction errors. RMSE can be interpreted as the average error the model makes in its predictions.
- **R^2** is calculated using the sum of squared error and the sum of the squared differences between the target values (y) and the mean of the target values (\bar{y}). The \bar{y} for the above Test data set is 92.5.

The MSE, RMSE and R² value and its interpretation for our trained model are as follows:

Metric -- >	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)	Coefficient of Determination (R ²)
Formula	$(1/N) * \sum (y_i - \hat{y}_i)^2$	$\text{sqrt}(\text{MSE}) = \text{sqrt}((1/N) * \sum (y_i - \hat{y}_i)^2)$	$1 - [\sum (y_i - \hat{y}_i)^2 / \sum (y_i - \bar{y})^2]$
Calculated Value	0.7538	0.8682	0.9759
Interpretation	Very low error	Predictions off by < 1 unit on average	Explains ~98% of variance.

Benefits

Revisiting ML fundamentals through linear regression delivers tangible technical and organizational advantages for enterprises seeking to scale AI and ML adoption responsibly and effectively. By anchoring ML practices in a transparent and mathematically explicit model, this whitepaper creates value across talent development, system reliability and operational maturity.

Accelerated talent development and knowledge transfer

By framing ML as a structured sequence of clearly defined steps – data preparation, model formulation, loss definition, optimization and evaluation – this whitepaper enables faster and more consistent upskilling of engineering teams. Linear regression serves as a minimal yet complete learning system that exposes all core mechanisms underlying modern ML architectures. This structured clarity supports stronger conceptual alignment, reduces knowledge fragmentation and improves the transferability of skills across projects and domains.

Improved model debugging and root-cause analysis

A deep understanding of loss functions, gradient-based optimization and evaluation metrics is essential for diagnosing failures in ML systems. By explicitly analyzing these components in the context of linear regression, this whitepaper equips practitioners with a systematic framework for understanding why models behave as they do. This structured reasoning capability strengthens root-cause analysis, enhances explainability and improves confidence when scaling models into production environments.

Discussion

Linear regression model performs effectively when the relationship between input variables (features) and the target variable (desired output) is linear. It is due to the underlying assumption of linearity in linear regression. The model assumes that the predicted output (target) \hat{y} can be expressed as a linear combination of the input variables (features). We saw in the Solutions section, a linear regression model having a single variable input (x) expressed as a linear combination of the input variable as $\hat{y} = w * x + b$. And multivariate linear regression is expressed as $\hat{y} = \sum w_i * x_i + b$ for input features $x_1, x_2, x_3, \dots, x_n$.

When the assumption of linearity holds, the linear regression model can provide accurate predictions and interpretable parameter estimates. However, when the true underlying relationship between input variables and the target variable is nonlinear, such as a polynomial relationship, or nonlinearly separable problems like the XOR classification task, then the linear model is incapable of capturing the true input-output mapping. While feature engineering can partially mitigate this limitation by transforming inputs into higher-dimensional spaces, this model complexity increase requires manual feature design. Modern approaches, such as artificial neural networks address, this limitation by learning nonlinear representations automatically through a layered architecture and nonlinear activation functions.

The primary goal of supervised ML models is generalization. It is true for linear regression models as well. **Generalization** is the ability of the model to perform well and make accurate predictions on unseen data (e.g., held-out validation data) after it has been trained on a known training dataset. A model generalizes well when the training error and test error are low. Linear regression generalizes well when its assumption of linearity holds. However, if the true relationship between input variables and the target variable is nonlinear, then the linear regression model exhibits poor generalization.

Conclusion

Linear regression, despite its simplicity, encapsulates the core principles that govern supervised ML models. Through this whitepaper, we examined model formulation, loss minimization, parameter optimization, model training and evaluation, that extend directly to more complex ML architectures.

Our exploration highlights three key principles. **First**, a model is a parameterized function, in our case, $\hat{y} = w*x + b$. **Second**, the loss function (MSE) serves as the crucial, objective measure of our model's performance by measuring how wrong or erroneous the model prediction (\hat{y}) is compared to the desired output or the target (y). **Finally**, the optimization algorithm (e.g., BGD) is the engine of ML. By using the gradient - the direction of steepest ascent on the loss function, the algorithm iteratively subtracts the gradient to refine the model's parameters and progressively minimizing the prediction error.

References

1. I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.
2. Jiang H. Machine Learning Fundamentals: A Concise Introduction. Cambridge University Press; 2021.
3. S. Ruder, "An Overview of Gradient Descent Optimization Algorithms," arXiv:1609.04747.
4. A. Ng, Machine Learning Course Notes, Stanford University.

Author information

Titu Doley



Titu is an Engineering Manager and AI/ML professional with deep expertise in system software development and test lifecycle engineering for consumer electronics platforms. His work centres on identifying high-impact engineering challenges that can be addressed through AI-driven solutions and applying machine learning techniques to enhance the efficiency, reliability and scalability of complex engineering workflows.

His current technical interests span deep learning, system-level analysis of machine learning pipelines and the application of reinforcement learning in intelligent and autonomous systems. With a strong focus on bridging theory and practice, Titu authors technical whitepapers on machine learning fundamentals to translate mathematical concepts into real-world system implementation, enabling engineers to build more robust and production-ready AI solutions.

HCLTech | Supercharging Progress™

HCLTech is a global technology company, home to more than 226,600 people across 60 countries, delivering industry-leading capabilities centered around AI, digital, engineering, cloud and software, powered by a broad portfolio of technology services and products. We work with clients across all major verticals, providing industry solutions for Financial Services, Manufacturing, Life Sciences and Healthcare, High Tech, Semiconductor, Telecom and Media, Retail and CPG, Mobility and Public Services. Consolidated revenues as of 12 months ending December 2025 totaled \$14.5 billion. To learn how we can supercharge progress for you, visit hcltech.com.

hcltech.com

