Stanford CS229: Machine Learning - Linear Regression and Gradient Descent | Lecture 2 (Autumn 2018)

Subject: Machine Learning
Instructor: N/A
Duration: 78:09

Overview

This lecture develops linear regression as a supervised learning method for predicting continuous outputs. It introduces a linear hypothesis and consistent notation for examples, features, parameters, and training sets; defines ordinary least-squares training through squared-error minimization; and presents batch gradient descent, stochastic gradient descent, and the normal equations as methods for obtaining model parameters. The final section reformulates least squares in matrix form and derives the normal equations using matrix derivatives and trace identities.

Prerequisites

- Basic single-variable and partial differentiation
- Vectors, matrices, transpose, and matrix multiplication
- The distinction between continuous regression outputs and classification outputs

Learning Outcomes

- Formulate linear regression as a supervised learning problem with a training set and a hypothesis.
- Represent a linear hypothesis using feature vectors, including an intercept feature.
- Define and interpret the least-squares cost function.
- Apply the batch gradient-descent update for linear regression and reason about the learning rate.
- Compare batch and stochastic gradient descent for small and large data sets.
- Express least squares in matrix form and state the normal equations.

Glossary

Term	Definition
Supervised learning	Learning from training examples that pair inputs with known outputs, with the objective of producing a hypothesis that predicts outputs for new inputs.

Term	Definition
Regression	A supervised learning task whose target variable is continuous, such as a house price or steering direction.
Training set	The collection of labelled examples used by the learning algorithm to choose model parameters.
Hypothesis	The prediction function output by a learning algorithm. In this lecture it is denoted by h or h_\theta.
Feature	An input variable used to make a prediction. Features are denoted by x_j.
Target variable	The output value to be predicted, denoted by y.
Parameter	A value chosen by the learning algorithm to determine a hypothesis. Linear-regression parameters are denoted by \theta.
Cost function	A scalar function measuring the training error of a parameter vector. The least-squares cost is denoted by J(\theta).
Learning rate	The positive scalar \alpha that controls the step size in gradient descent.
Batch gradient descent	Gradient descent in which each parameter update uses the sum of contributions from the entire training set.
Stochastic gradient descent	Gradient descent in which each update uses one training example, producing inexpensive but noisy updates.
Design matrix	The matrix X formed by stacking the transposes of all training feature vectors as rows.
Trace	For a square matrix, the sum of its diagonal entries, denoted by \operatorname{tr}(A).
Normal equations	The linear system X^T X\theta=X^T y obtained by setting the gradient of the least-squares cost to zero.

Introduction to linear regression and supervised learning

Difficulty: *Beginner* ***Time:*** *00:03 - 04:34*

Summary

Linear regression is introduced as a simple supervised-learning regression algorithm. A housing-price data set illustrates the general workflow: a training set contains house sizes and known prices; a learning algorithm uses these examples to construct a hypothesis; and the hypothesis receives the features of a previously unseen house and returns an estimated price. The central design decision is the representation of this hypothesis, which is developed in the following topic.

Learning Objectives

- Distinguish supervised regression from classification in the context of a prediction task.
- Describe the roles of a training set, learning algorithm, and hypothesis.

Key Concepts

Supervised-learning workflow

Importance: High

A supervised-learning problem supplies input-output pairs. The learning algorithm receives a training set and produces a hypothesis h. The resulting hypothesis is then used to map inputs from new cases to predicted outputs.

Related Concepts: training set, hypothesis, target variable

Regression target

Importance: High

The housing-price example is a regression problem because the desired output, price, is continuous. This contrasts with classification, where the output is a discrete category.

Related Concepts: supervised learning, linear regression

Housing-price example

Importance: Medium

Each observed house contributes a size-price pair to the training set. Plotting these pairs makes the prediction problem geometric: linear regression will fit a straight line relating the input size to the predicted price.

Related Concepts: training set, linear hypothesis

Worked Examples

Housing-price prediction task

Problem

Given recorded house sizes and their asking prices, estimate the price of a new house from its size.

Solution

Treat each observed size-price pair as a labelled training example. A learning algorithm fits a hypothesis to these examples; the hypothesis then maps the size of a new house to an estimated continuous price.

Visual Explanations

From table to prediction function

The training data can be represented both as a two-column table of house sizes and prices and as points in a coordinate system. The learning pipeline maps this training set through a learning algorithm to a hypothesis h, which accepts a new input and produces a predicted price.

Common Mistakes

Common Mistake

Treating every supervised-learning task as classification.

Correction

Use regression when the required output is continuous, as in price prediction; classification is reserved for discrete outputs.

Key Takeaways

- Supervised learning learns a predictive function from labelled examples.
- A training set is used to construct a hypothesis rather than directly serving as the final predictor.
- Linear regression is a supervised regression method that will fit a straight-line prediction rule.

Selected Frames

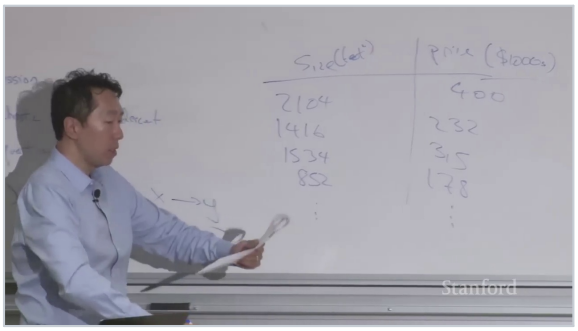

Figure 1 • *02:45*

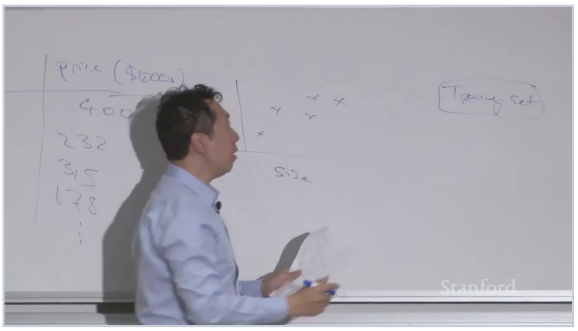

Figure 2 • *03:11*

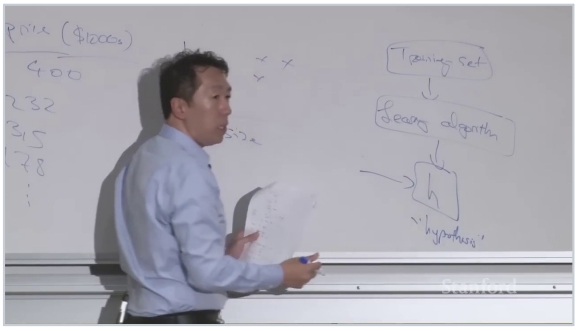

Figure 3 • *03:38*

Linear hypothesis representation and training-data notation

Difficulty: *Beginner* ***Time:*** *04:34 - 12:58*

Summary

Linear regression represents the hypothesis as an affine function of the input features. For house size and number of bedrooms, the prediction is a weighted sum of an intercept and the two feature values. Introducing the dummy feature $x_0=1$ allows this expression to be written as one summation. The topic also establishes the notation used throughout the lecture: m is the number of training examples, n is the number of original features, θ denotes parameters, and (x^{(i)},y^{(i)}) denotes the i-th labelled example.

Learning Objectives

- Write a linear-regression hypothesis for one or multiple input features.
- Use the intercept-feature convention x_0=1 to express the hypothesis compactly.
- Interpret the indexing notation for training examples, features, and parameters.

Key Concepts

Linear hypothesis

Importance: High

For a single feature x, linear regression predicts an output as an intercept plus a coefficient times x. With multiple features, each feature has its own coefficient. Although the intercept makes this technically an affine function, it is conventionally called linear regression in this setting.

Related Concepts: parameters, features, intercept

Intercept feature convention

Importance: High

Defining $x_0=1$ converts the intercept θ_0 into the same form as all other weighted feature terms. It permits the hypothesis to be expressed as one summation from j=0 to n.

Related Concepts: linear hypothesis, feature vector, parameter vector

Example and feature indices

Importance: High

The parenthesized superscript in x^{(i)} and y^{(i)} indexes the i-th training example; it is not exponentiation. The subscript x_j indexes a feature within an example. Thus x_j^{(i)} is feature j of training example i.

Related Concepts: training set, m, n

Parameter learning

Importance: High

The vector θ contains the parameters selected by the learning algorithm. Its purpose is to make the hypothesis produce accurate predictions on the available examples.

Related Concepts: cost function, gradient descent

Equations

Two-feature linear hypothesis

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

For the housing example, x_1 is house size and x_2 is the number of bedrooms. The parameters θ_0, θ_1, and θ_2 determine the predicted price.

Variables

h(x)	predicted output
x_1	house size
x_2	number of bedrooms
θ_0	intercept parameter
θ_1,θ_2	feature parameters

Compact linear hypothesis

$$h(x) = \sum_{j=0}^{n} \theta_j x_j$$

With x_0=1, the intercept is included in the same weighted sum as the remaining features. The hypothesis and parameter vector are both n+1 dimensional under this convention.

Variables

n	number of original input features
x_0	dummy feature fixed at 1
θ_j	parameter associated with feature x_j

Worked Examples

Feature indexing in the housing data

Problem

Interpret the notation x_1^{(1)} when the first recorded house has size 2104.

Solution

The parenthesized superscript selects the first training example, while the subscript 1 selects the first feature. Therefore x_1^{(1)}=2104 when feature 1 is house size.

Visual Explanations

Feature and parameter vectors

For two original features, the augmented feature vector is formed from x_0, x_1, and x_2, while the parameter vector contains θ_0, θ_1, and θ_2. The dummy first component permits the intercept to be handled uniformly.

Common Mistakes

Common Mistake

Reading the parenthesized superscript (i) as a power.

Correction

It is an index identifying a row, or training example, in the data set.

Common Mistake

Counting x_0 as an original measured feature.

Correction

x_0 is introduced solely as a dummy feature equal to 1 so that the intercept can be included in summation notation.

Key Takeaways

- A linear hypothesis is a weighted sum of input features plus an intercept.
- The convention $x_0=1$ yields the compact expression h(x)=Σ_{j=0}^{n}θ_jx_j.
- m indexes the number of examples, while n indexes the number of original features.

Selected Frames

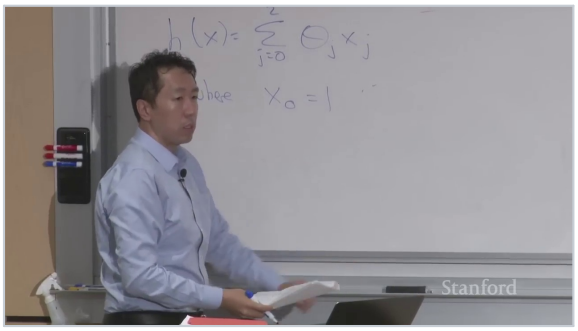

Figure 1 • *07:38*

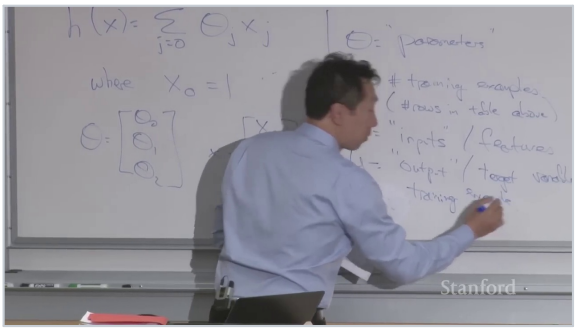

Figure 2 • *10:17*

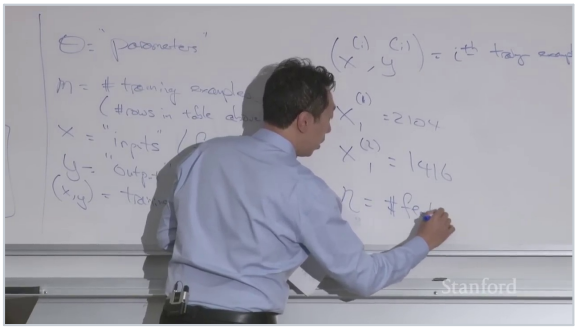

Figure 3 • *11:45*

Least-squares cost function

Difficulty: *Intermediate* ***Time:*** *12:58 - 17:32*

Summary

Training linear regression means choosing parameters that make predictions close to the known targets for the training examples. The hypothesis is written as h_θ(x) when it is useful to emphasize its dependence on both the input and the parameter vector. Ordinary least squares defines a cost function as one-half of the sum of squared prediction errors and chooses the parameters that minimize this quantity.

Learning Objectives

- Define the prediction error for an individual training example.
- Write and interpret the ordinary least-squares cost function.
- Explain the role of the factor one-half in the cost function.

Key Concepts

Parameter-dependent hypothesis

Importance: Medium

The notation h_θ(x) emphasizes that a prediction depends both on an input x and on the current parameters θ. The abbreviated notation h(x) is used when the parameter dependence is clear from context.

Related Concepts: hypothesis, parameters

Squared prediction error

Importance: High

For example i, h_θ(x^{(i)})-y^{(i)} is the difference between the predicted and known target value. Squaring this difference yields a nonnegative contribution to the total training cost.

Related Concepts: least squares, cost function

Ordinary least squares

Importance: High

Ordinary least squares chooses parameters by minimizing the aggregate squared error over all m training examples. It is the training objective used for linear regression in this lecture.

Related Concepts: cost function, gradient descent, normal equations

Equations

Least-squares cost function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

The cost J(θ) aggregates squared prediction errors over the full training set. Minimizing it selects parameters whose predictions are close to the known targets on these examples. The factor one-half does not change which θ minimizes the expression; it simplifies later differentiation by cancelling the factor 2 produced by differentiating a square.

Variables

m	number of training examples
x^{(i)}	features of example i
y^{(i)}	target of example i
h_θ(x^{(i)})	prediction for example i
J(θ)	least-squares cost

Visual Explanations

Cost as aggregate fit

The training objective formalizes the requirement that the predicted price be close to the known price for every observed house. Each example contributes its own squared discrepancy, and the total cost adds these discrepancies.

Common Mistakes

Common Mistake

Assuming the factor one-half changes the optimal parameters.

Correction

Multiplying the objective by the positive constant one-half does not alter its minimizer; it is included to simplify derivative calculations.

Common Mistake

Confusing a prediction with its error.

Correction

h_\theta(x^{(i)}) is the prediction, whereas h_\theta(x^{(i)})-y^{(i)} is the prediction error.

Key Takeaways

- Least-squares training minimizes the summed squared discrepancies between predictions and targets.
- The objective is a function of the parameters θ, not a quantity fixed independently of the model.
- The squared-error choice is stated to be justified later through generalized linear models and its connection to a Gaussian setting.

Selected Frames

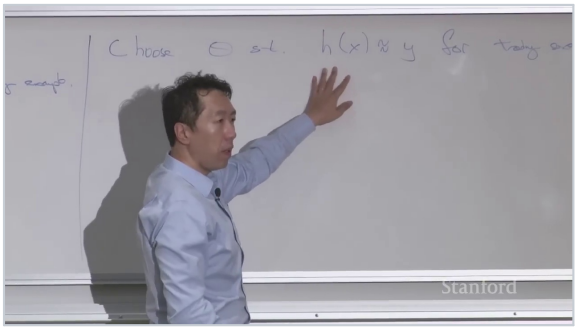

Figure 1 • *13:51*

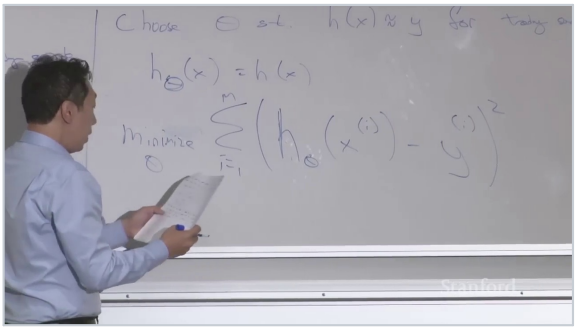

Figure 2 • *15:55*

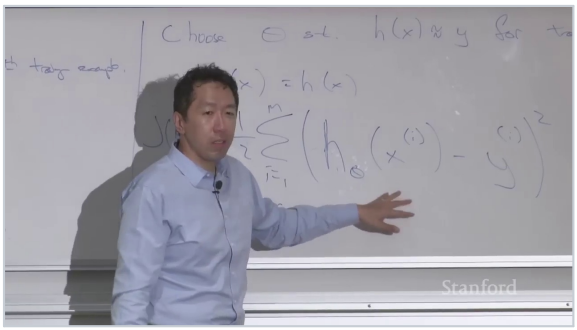

Figure 3 • *16:43*

Batch gradient descent for linear regression

Difficulty: *Intermediate* ***Time:*** *17:42 - 40:57*

Summary

Batch gradient descent is an iterative method for minimizing the least-squares cost. Starting from an initial parameter vector, it repeatedly moves parameters in the direction opposite to the gradient. For linear regression, each batch update uses all training examples. The lecture derives the gradient contribution of an example, explains simultaneous updates for all parameters, and shows that the least-squares cost has a bowl-shaped quadratic form with no non-global local minimum. The learning rate controls the trade-off between overly large, unstable steps and slow progress.

Learning Objectives

- State the batch gradient-descent update and explain its negative-gradient direction.
- Derive the single-example derivative contribution for a linear hypothesis.
- Explain the effects of an excessively large or small learning rate.
- Relate the geometry of a quadratic least-squares cost to convergence toward its global minimum.

Key Concepts

Gradient descent

Importance: High

Gradient descent starts from an initial parameter vector and repeatedly changes the parameters to reduce J(θ). Geometrically, the gradient identifies the direction of greatest local increase, so subtracting a scaled gradient moves most steeply downhill locally.

Related Concepts: gradient, learning rate, cost function

Assignment notation

Importance: Medium

The symbol := denotes assignment: the expression on the right replaces the current value on the left. It differs from =, which asserts equality as a mathematical statement.

Related Concepts: gradient-descent update

Batch update

Importance: High

A batch update calculates the derivative using the entire training set before changing any parameter. Every parameter component is updated using the same pre-update parameter vector; the updates are carried out simultaneously.

Related Concepts: batch gradient descent, least-squares gradient

Learning rate selection

Importance: High

The learning rate α scales the gradient step. A value that is too large can overshoot the minimum and cause the cost to increase; a value that is too small requires many iterations. The lecture recommends trying several values, often on an exponentially spaced scale, and monitoring how efficiently J(θ) decreases.

Related Concepts: convergence, feature scaling

Least-squares geometry

Importance: High

For linear regression with squared error, J(θ) is a quadratic bowl. Its contour lines are ellipses, and the only local optimum is also the global optimum. This special structure avoids the distinct local minima illustrated by a more general nonconvex surface.

Related Concepts: global minimum, contours, linear regression

Equations

General gradient-descent update

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

For each j from 0 through n, the update subtracts the j-th partial derivative of the cost, scaled by the learning rate. The negative sign is essential because the gradient points uphill; subtraction moves toward lower cost.

Variables

θ_j	parameter component j
α	learning rate
J(θ)	cost function
\frac{}{θ_j}J(θ)	partial derivative with respect to parameter j

Single-example derivative contribution

$$\frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y\right)^2 = \left(h_\theta(x) - y\right) x_j$$

Differentiating the squared error first produces the residual h_θ(x)-y. Since h_θ(x)=Σ_{k=0}^{n}θ_kx_k, differentiating with respect to θ_j leaves x_j; all other weighted-feature terms do not depend on θ_j.

Variables

x_j	feature j of the example
h_θ(x)-y	prediction residual

Batch least-squares gradient

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

The cost is a sum over examples, so its derivative is the sum of the individual derivative contributions. Each term combines the residual of example i with feature j of that same example.

Variables

m	number of examples
x_j^{(i)}	feature j of example i
y^{(i)}	target of example i

Batch gradient-descent update for linear regression

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

This is the linear-regression batch update obtained by substituting the least-squares derivative into gradient descent. It is repeated until convergence, with the update performed for every j from 0 to n.

Variables

θ_j	parameter being updated
α	learning rate
i	training-example index

Algorithms

Batch gradient descent for least squares

Intuition

Each iteration examines the total training error over the complete data set and moves the parameter vector downhill on the least-squares cost surface.

Steps

1. Initialize θ, for example as the all-zero vector.
2. Repeat until convergence.

3 For every parameter index j=0,1,\ldots,n, compute the full-training-set gradient contribution $\Sigma_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$.

4 Simultaneously assign each θ_j its previous value minus α times that contribution.

5 Monitor $J(\theta)$; persistent increases are a strong indication that α is too large.

Complexity: Each update requires a scan through all m training examples.

Worked Examples

Why the gradient is subtracted

Problem

Explain the sign of the update using $J(\theta)=\theta^2$ at a positive value of \theta.

Solution

At a positive \theta, the derivative is positive. Reducing J requires moving \theta toward zero, hence decreasing \theta. Subtracting a positive multiple of the derivative performs this decrease; adding it would move uphill.

Visual Explanations

Surface and contour views of gradient descent

A three-dimensional cost surface uses parameter values as horizontal coordinates and $J(\theta)$ as height. Gradient descent follows a sequence of local downhill steps. Horizontal slices of the quadratic bowl form nested elliptical contours; the steepest-descent direction is orthogonal to a contour at the current point.

Fitted-line view of optimization

When the parameters are initialized at zero in a one-feature model, the hypothesis predicts zero for every input and is therefore a horizontal line. Successive gradient-descent iterations alter the intercept and slope, producing lines that increasingly fit the housing-price points.

Common Mistakes

Common Mistake

Adding the gradient in a minimization update.

Correction

Adding a positive multiple of the gradient moves in the direction of increasing cost. Minimization requires subtracting the gradient.

Common Mistake

Using a large learning rate because it produces larger immediate steps.

Correction

A learning rate that is too large can overshoot the minimum and make J(\theta) increase.

Common Mistake

Interpreting the generic multi-minimum surface as the least-squares linear-regression objective.

Correction

The lecture distinguishes the generic illustration from linear least squares, whose squared-error cost is a quadratic bowl with no separate local minima.

Key Takeaways

- Batch gradient descent repeatedly subtracts a learning-rate-scaled gradient from every parameter.
- For linear regression, the batch gradient sums residual-times-feature terms over all examples.
- The least-squares cost has a single global minimum; learning-rate choice determines the efficiency and stability of reaching it.
- Feature scaling to a range such as approximately -1 to 1 supports practical learning-rate experimentation.

Selected Frames

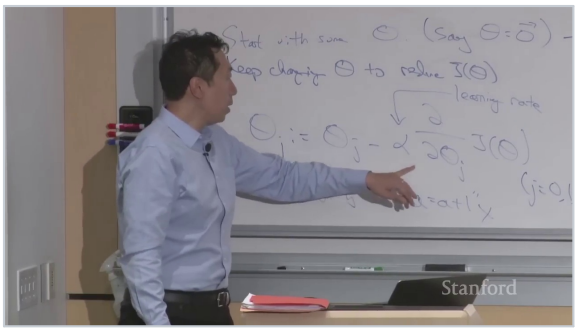

Figure 1 • *25:11*

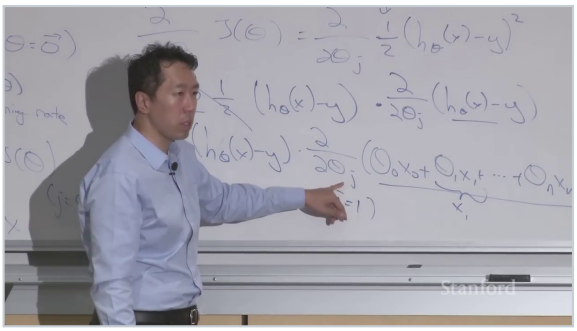

Figure 2 • *29:59*

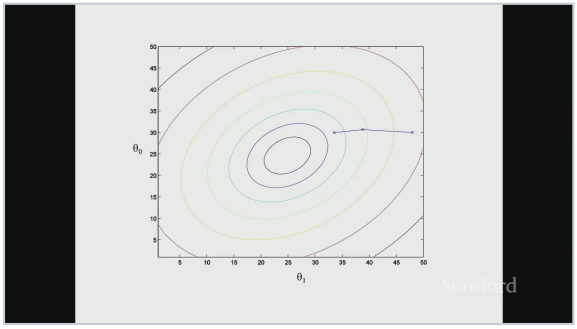

Figure 3 • *35:37*

Stochastic gradient descent and practical comparison with batch methods

Difficulty: *Intermediate* ***Time:*** *40:57 - 52:34*

Summary

Batch gradient descent performs one update only after processing all m examples, which can be impractically slow for very large data sets. Stochastic gradient descent instead updates parameters after each individual example. Its path toward the minimum is noisy and generally oscillates rather than converging exactly, but it can make progress much sooner because each update is inexpensive. For small data sets, batch gradient descent is preferred for its simpler, stable behavior; for large data sets, stochastic or closely related methods are typically used. Decreasing the learning rate over time reduces stochastic oscillations.

Learning Objectives

- Explain why batch gradient descent is expensive on very large data sets.
- State the stochastic gradient-descent update for linear regression.
- Compare the trajectory and convergence behavior of batch and stochastic gradient descent.
- Identify practical stopping and learning-rate strategies described for stochastic gradient descent.

Key Concepts

Meaning of batch

Importance: High

In batch gradient descent, the entire training set is treated as one batch. The gradient sum must be computed across all examples before even one parameter update is made.

Related Concepts: batch gradient descent, large data sets

Stochastic updates

Importance: High

Stochastic gradient descent processes examples one at a time. It adjusts parameters to improve the fit for the current example, then immediately proceeds to the next example.

Related Concepts: online updates, noisy trajectory

Noisy convergence behavior

Importance: High

Because each update uses only one example, individual steps are not necessarily the direction that most directly reduces the full cost. The trajectory is noisy and tends to oscillate around a good region rather than settling exactly at the minimum.

Related Concepts: learning-rate decay, global minimum

Learning-rate decay

Importance: Medium

Rather than switching to batch gradient descent, the lecture notes that a common practical strategy is to continue stochastic gradient descent while gradually decreasing α. Smaller steps reduce the size of the final oscillations.

Related Concepts: stochastic gradient descent, convergence

Method selection by data size

Importance: High

When a data set is small enough that full-data updates are inexpensive, batch gradient descent is preferred because it avoids the extra behavior of oscillating stochastic updates. When full passes are prohibitively slow, stochastic gradient descent is more practical.

Related Concepts: batch gradient descent, large-scale learning

Equations

Stochastic gradient-descent update

$$\theta_j := \theta_j - \alpha \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

For the current example i, this update uses only that example's residual and feature value rather than a sum over all m examples. For each i from 1 to m, the update is applied for every parameter index j.

Variables

i	current training-example index
θ_j	parameter component j
α	learning rate
x_j^{(i)}	feature j of example i
y^{(i)}	target of example i

Algorithms

Stochastic gradient descent for least squares

Intuition

Instead of waiting to compute the full-data gradient, update immediately after examining each training example. Each step is noisy but cheap, enabling rapid progress when the data set is large.

Steps

1. Repeat over the training data.
2. For i=1,2,\ldots,m, select the current labelled example (x^{(i)},y^{(i)}).
3. For each j=0,1,\ldots,n, update θ_j using the residual of only example i.
4. Monitor J(θ) over time; stop when it no longer appears to decrease.
5. Optionally decrease the learning rate over time to reduce the magnitude of late-stage oscillations.

Complexity: Each individual update uses one training example; a full pass processes m examples.

Visual Explanations

Batch and stochastic trajectories on contours

Batch gradient descent follows directions computed from the full objective and moves smoothly toward the global minimum of the least-squares contours. Stochastic gradient descent follows a more irregular path because each move reflects only one example, but its average direction still progresses toward the minimum.

Common Mistakes

Common Mistake

Expecting stochastic gradient descent to stop exactly at the global minimum with a fixed learning rate.

Correction

Its one-example updates cause persistent oscillation; decreasing the learning rate reduces, but does not guarantee elimination of, this behavior.

Common Mistake

Assuming batch gradient descent is always preferable because it uses the complete data set per update.

Correction

For very large data sets, the cost of a single full-data update can be prohibitive, making stochastic updates more useful in practice.

Common Mistake

Scaling the stochastic learning rate by 1/m solely to match a batch update.

Correction

The lecture notes that such a choice would make stochastic progress as slow as batch gradient descent; in practice its learning rate is usually larger.

Key Takeaways

- Batch gradient descent is costly because every update requires a complete pass through the training set.
- Stochastic gradient descent updates after each example and is therefore well suited to large data sets.
- Stochastic trajectories are noisy and oscillatory, but reducing the learning rate over time decreases the oscillation scale.
- Monitor J(θ) during training and stop when it no longer decreases appreciably.

Further Reading

Mini-batch gradient descent

A related method mentioned in the lecture uses a small group of examples per update rather than one example or the entire training set.

Selected Frames

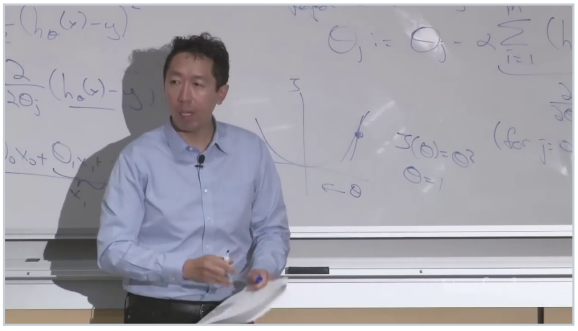

Figure 1 • *41:05*

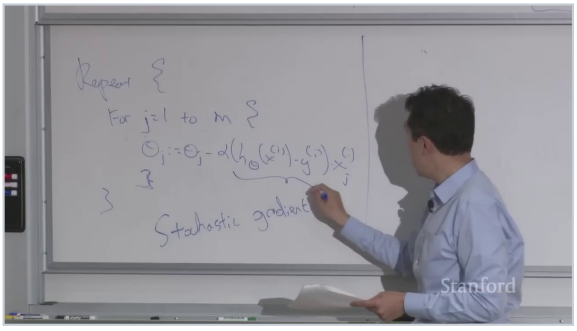

Figure 2 • *45:23*

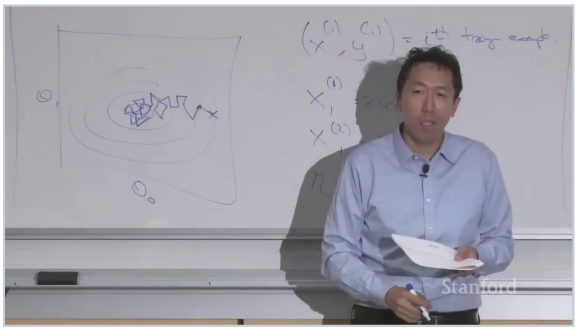

Figure 3 • *47:07*

Normal equations and matrix derivation for linear regression

Difficulty: *Advanced* ***Time:*** *52:47 - 78:13*

Summary

The normal equations provide a non-iterative solution for the least-squares linear-regression parameters. The derivation introduces vector and matrix derivatives, the trace operator, and a matrix representation of the training data. Stacking training examples into the design matrix X makes Xθ the vector of predictions and makes the least-squares cost a squared norm. Differentiating this matrix expression and setting the derivative to zero yields X^T Xθ=X^T y. When the inverse is usable, the resulting closed-form expression gives the global least-squares optimum. This method is specific to linear regression; later algorithms require iterative methods such as gradient descent.

Learning Objectives

- Define vector and matrix derivatives as arrays of entrywise partial derivatives.
- State the trace definition and matrix identities used in the derivation.
- Construct the design matrix and vectorized target representation.
- Derive and state the normal equations and their closed-form solution.
- Explain the limitation caused by redundant, linearly dependent features.

Key Concepts

Non-iterative least-squares solution

Importance: High

Unlike gradient descent, the normal-equation method computes the optimal linear-regression parameters directly through matrix operations. It applies only to the exact linear-regression setting discussed here, not generally to the later learning algorithms mentioned in the lecture.

Related Concepts: normal equations, gradient descent, least squares

Vector and matrix derivatives

Importance: High

For a scalar function of a vector, the derivative with respect to that vector is a column vector of partial derivatives. For a scalar function f(A) of a matrix A, the derivative with respect to A is a matrix of the same dimensions, whose entries are the partial derivatives with respect to the corresponding entries of A.

Related Concepts: gradient, matrix calculus

Trace operator

Importance: High

The trace of a square matrix is the sum of its diagonal entries. Trace identities make it possible to manipulate matrix expressions into forms whose derivatives can be computed compactly.

Related Concepts: matrix derivatives, normal-equation derivation

Design matrix

Importance: High

The design matrix X stacks the transpose of each feature vector as a row. Consequently, Xθ is a column vector whose i-th component is x^{(i)T}θ, which equals the prediction h_θ(x^{(i)}).

Related Concepts: feature vectors, prediction vector, least-squares objective

Redundant features

Importance: High

If the matrix required for the closed-form expression is non-invertible, the lecture identifies linearly dependent or repeated features as a usual cause. A pseudoinverse can provide an answer, but examining and removing redundant features is presented as the more appropriate response.

Related Concepts: matrix inverse, feature design

Equations

Gradient with respect to a parameter vector

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix}$$

Because θ is a vector, its derivative is formed by stacking the partial derivatives of J with respect to its components. The optimum is found by setting this gradient vector equal to the zero vector.

Variables

θ	parameter vector
J(θ)	scalar cost function

Matrix derivative definition

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \frac{\partial f}{\partial A_{12}} \\ \frac{\partial f}{\partial A_{21}} & \frac{\partial f}{\partial A_{22}} \end{bmatrix}$$

For a 2-by-2 matrix A, the derivative of a scalar-valued function f(A) is a 2-by-2 matrix containing one partial derivative for each matrix entry. The lecture uses this definition as the matrix analogue of the vector derivative.

Variables

A_{ij}	entry in row i and column j of A
f(A)	scalar-valued function of A

Trace definition

$$\operatorname{tr}(A) = \sum_i A_{ii}$$

The trace is defined for a square matrix as the sum of diagonal entries. It is used to state compact matrix-derivative identities.

Variables

A_{ii}	i-th diagonal entry of square matrix A

Trace identities

$$\operatorname{tr}(AB) = \operatorname{tr}(BA)$$

The trace of a product is unchanged when the two factors are cyclically exchanged. The lecture also states the corresponding cyclic identity for three factors.

Variables

A,B	matrices with compatible dimensions

Three-factor cyclic trace identity

$$\operatorname{tr}(ABC) = \operatorname{tr}(CAB)$$

Within a trace, a product can be cyclically reordered by moving the final factor to the front. This identity is used in matrix-calculus manipulations.

Variables

A,B,C	matrices with compatible dimensions

Trace derivative identity

$$\nabla_A \operatorname{tr}(AB) = B^T$$

For fixed B, differentiating the trace of AB with respect to A gives B transpose. This is one of the stated identities used to shorten the normal-equation derivation.

Variables

A	matrix variable
B	fixed compatible matrix

Quadratic trace derivative identity

$$\nabla_A \operatorname{tr}(AA^T C) = CA + C^T A$$

This stated matrix derivative identity is the matrix analogue of differentiating a quadratic expression. It supports differentiating the quadratic least-squares objective.

Variables

A	matrix variable
C	compatible matrix

Design matrix

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix}$$

Each row of X is one training feature vector transposed. If each augmented feature vector has n+1 components, X has m rows and n+1 columns.

Variables

m	number of training examples
x^{(i)}	augmented feature vector for example i

Target vector

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

The target vector stacks the known outputs of all training examples in the same order as the rows of X.

Variables

y^{(i)}	target output of example i

Vector of predictions

$$X\theta = \begin{bmatrix} \left(x^{(1)}\right)^T \theta \\ \left(x^{(2)}\right)^T \theta \\ \vdots \\ \left(x^{(m)}\right)^T \theta \end{bmatrix}$$

The i-th entry of Xθ is the linear prediction for example i. Thus Xθ-y is the vector of prediction errors across the training set.

Variables

X	design matrix
θ	parameter vector
y	target vector

Matrix form of least-squares cost

$$J(\theta) = \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

The vector Xθ-y contains every residual. Multiplying this vector transposed by itself sums the squares of its entries, reproducing the ordinary least-squares cost.

Variables

Xθ-y	residual vector
J(θ)	least-squares cost

Least-squares gradient in matrix form

$$\nabla_\theta J(\theta) = X^T X\theta - X^T y$$

Expanding the quadratic matrix form and applying the stated derivative identities yields this gradient. Setting it to zero gives the condition for the least-squares optimum.

Variables

X^TX	feature cross-product matrix
X^Ty	feature-target product vector

Normal equations

$$X^T X\theta = X^T y$$

At an optimum, the gradient of the least-squares cost is set to zero. Rearranging the resulting gradient equation gives the normal equations.

Variables

X	design matrix
θ	optimal parameter vector

y	target vector

Closed-form normal-equation solution

$$\theta = (X^T X)^{-1} X^T y$$

When X^TX is invertible, multiplying the normal equations by its inverse gives the parameter vector at the global least-squares minimum. If the relevant matrix is non-invertible, the lecture notes the use of a pseudoinverse and highlights redundant features as a likely underlying issue.

Variables

(X^TX)⁻¹	inverse of X^TX, when it exists
θ	least-squares parameter vector

Algorithms

Normal-equation solution for linear regression

Intuition

Least squares is a quadratic function of the parameters. Rather than descending iteratively, differentiate the vectorized objective, set its gradient to zero, and solve the resulting linear system.

Steps

1. Form the augmented feature vector for each example, including $x_0=1$.
2. Stack the transposed feature vectors as rows to form X, and stack targets into y.
3. Form the normal equations X^TXθ=X^Ty by setting _θ J(θ)=0.
4. When X^TX is invertible, compute θ=(X^TX)⁻¹X^Ty.
5. If invertibility fails, investigate linearly dependent or repeated features; the lecture notes that a pseudoinverse can also be used.

Complexity: The lecture characterizes this as a direct matrix-operation method but does not provide a numerical complexity bound.

Worked Examples

Matrix-function derivative example

Problem

For a 2-by-2 matrix A, let f(A)=A_{11}+A_{12}^2. Determine the structure of \nabla_A f(A).

Solution

Differentiate entry by entry. The derivative with respect to A_{11} is 1, with respect to A_{12} is 2A_{12}, and with respect to A_{21} and A_{22} is 0 because f does not depend on them. Thus the matrix derivative has the same 2-by-2 shape as A and contains these four partial derivatives in corresponding positions.

Visual Explanations

Design matrix as stacked examples

The design matrix arranges examples by rows rather than treating them separately. Matrix-vector multiplication by θ therefore computes all linear predictions simultaneously, one prediction per row.

Residual vector and squared norm

Subtracting the target vector from the prediction vector gives one residual for each training example. The product (Xθ-y)^T(Xθ-y) is the sum of squares of these residuals, directly matching the original least-squares objective.

Common Mistakes

Common Mistake

Applying the normal equations as a general replacement for gradient descent.

Correction

The lecture states that this direct solution is specific to linear regression; other algorithms discussed later require iterative optimization.

Common Mistake

Assuming a non-invertible matrix is merely a numerical detail without examining the features.

Correction

The lecture identifies redundant, linearly dependent features as a usual cause and recommends determining which features are repeated.

Common Mistake

Confusing X\theta with a single prediction.

Correction

X\theta is the full vector of predictions for all m training examples; each component corresponds to one row of the design matrix.

Key Takeaways

- Matrix notation compresses all predictions and residuals into Xθ and Xθ-y.
- Setting the least-squares gradient to zero yields the normal equations X^TXθ=X^Ty.
- When the required inverse exists, the normal equations provide the global optimum without iterative gradient-descent steps.
- The direct normal-equation method is limited to linear regression; gradient-descent methods remain necessary more broadly.

Selected Frames

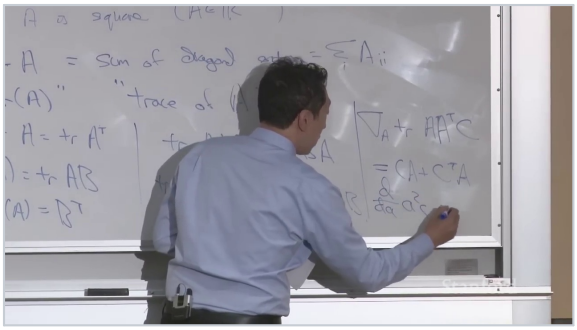

Figure 1 • *67:17*

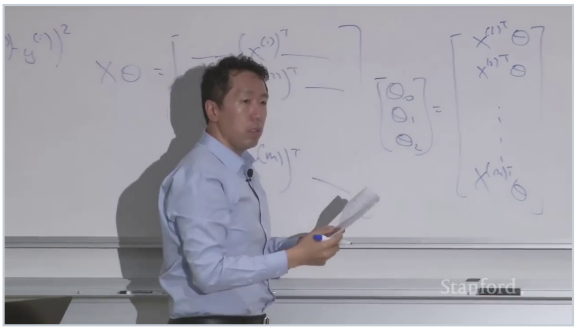

Figure 2 • *70:01*

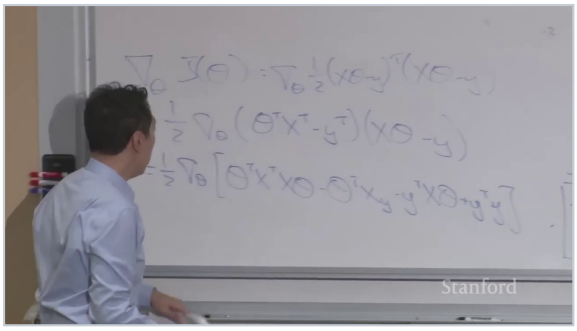

Figure 3 • *74:30*

Executive Summary

Linear regression is formulated as supervised learning with continuous targets. A linear hypothesis combines an intercept and weighted input features, and ordinary least squares chooses the parameters that minimize the summed squared training residuals. Batch gradient descent minimizes this objective using full-data gradients; it is stable for manageable data sets but expensive when each pass through the data is costly. Stochastic gradient descent uses one-example updates, making rapid progress on large data sets at the cost of noisy, oscillatory parameter trajectories; learning-rate decay reduces these oscillations. For linear regression alone, the least-squares objective can also be written in matrix form and solved directly through the normal equations. The design matrix representation yields the condition X^TX\theta=X^Ty and, when invertible, the closed-form solution \theta=(X^TX)^{-1}X^Ty.