CPEN 455: Deep Learning

Lecture 2 : Linear Models

Renjie Liao

University of British Columbia

Winter, Term 1, 2026

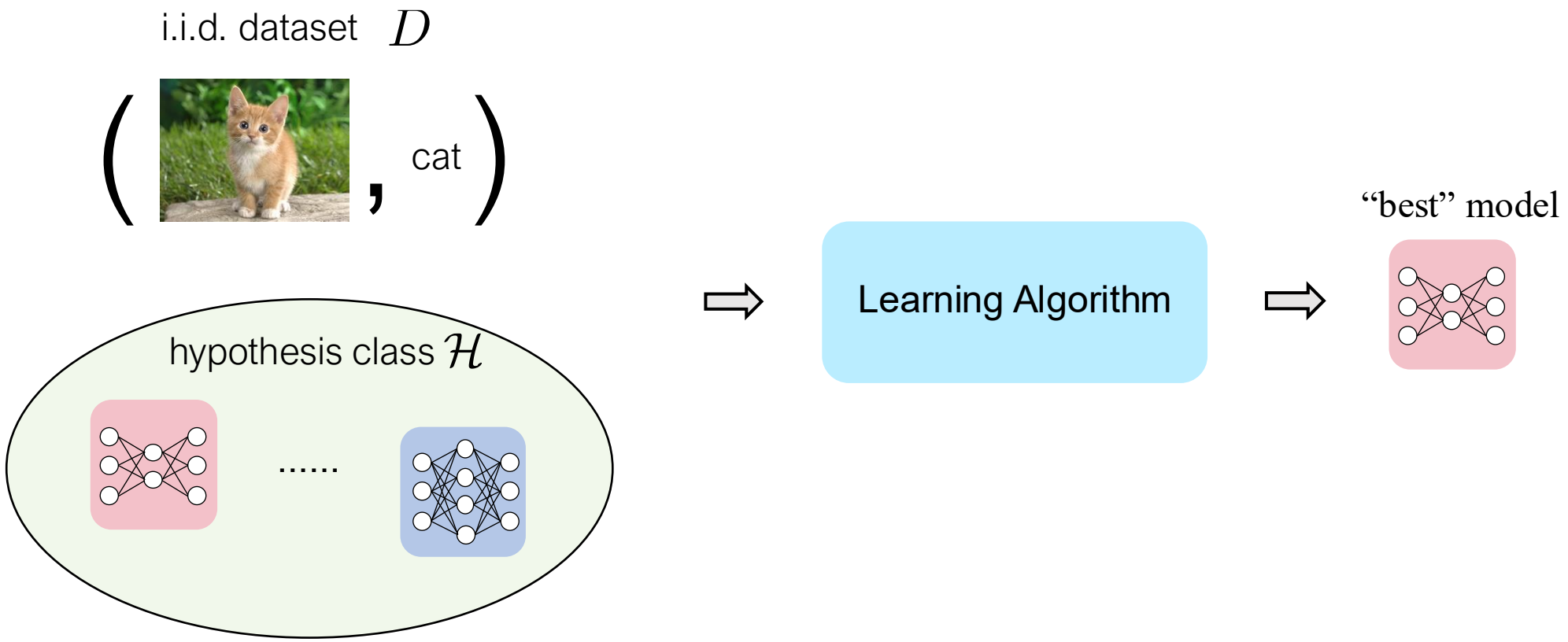
Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

Outline

- **Statistical Learning Setup**
- Linear Regression
  - Model and loss
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

Statistical Learning Setup



i.i.d. = independent and identically distributed: examples are sampled independently from the same distribution.
An input and its label may still be dependent.

Statistical Learning Setup

- Problem Specification
  Identify the input, output, and target task.
- Model Design
  Choose the family of prediction functions.
- Loss Design
  Define how prediction errors are measured.
- Prediction/Inference Algorithm
  Compute a prediction using a fitted model.
- Learning/Training Algorithm
  Fit model parameters using the training data.
- Validation/Testing
  Select hyperparameters and evaluate performance on new data.

Statistical Learning Setup

- Data, model, and loss

We observe a training dataset sampled from an unknown distribution.

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}, \qquad (x_n, y_n) \overset{\text{i.i.d.}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

A model predicts the output. A loss measures how well the prediction matches the label.

$$\hat{y}_n = f(x_n, w), \qquad \ell(\hat{y}_n, y_n)$$

Parameters are learned from training data. Hyperparameters are selected using validation data.

Statistical Learning Setup

- Training loss and expected loss

We want good predictions on new data from the same distribution.

$$\underbrace{\mathbb{E}_{(x,y)\sim\mathbb{P}_{\text{data}}}[\ell(f(x,w),y)]}_{\text{expected loss / risk}} \qquad \underbrace{\frac{1}{N}\sum_{n=1}^{N}\ell(f(x_n,w),y_n)}_{\text{training loss / empirical risk}}$$

Since the data distribution is unknown, we minimize the average training loss.

$$\min_{w} \frac{1}{N}\sum_{n=1}^{N}\ell(f(x_n,w),y_n)$$

This is empirical risk minimization (ERM) [1]. Low training loss alone does not guarantee generalization.

Statistical Learning Setup

- Learning/Training Algorithm

  An iterative learning algorithm fits model parameters from the training data.

$$\widehat{w} = \mathcal{A}(\mathcal{D}, w_0, \lambda)$$

$$\mathcal{D} : \text{training data}, \quad w_0 : \text{initialization}, \quad \lambda : \text{hyperparameters}$$

- Prediction/Inference Algorithm

  Use the fitted model to predict the output for a new input.

$$\widehat{y} = f(x, \widehat{w})$$

The loss defines what to optimize. The learning algorithm determines how to optimize it.

We will instantiate these components with linear regression, using gradient descent as one learning algorithm.

Outline

- Statistical Learning Setup
- **Linear Regression**
  - **Model and loss**
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

Linear Regression

- Problem Specification (Scalar-Output Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

Linear Regression

- Problem Specification (Scalar-Output Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

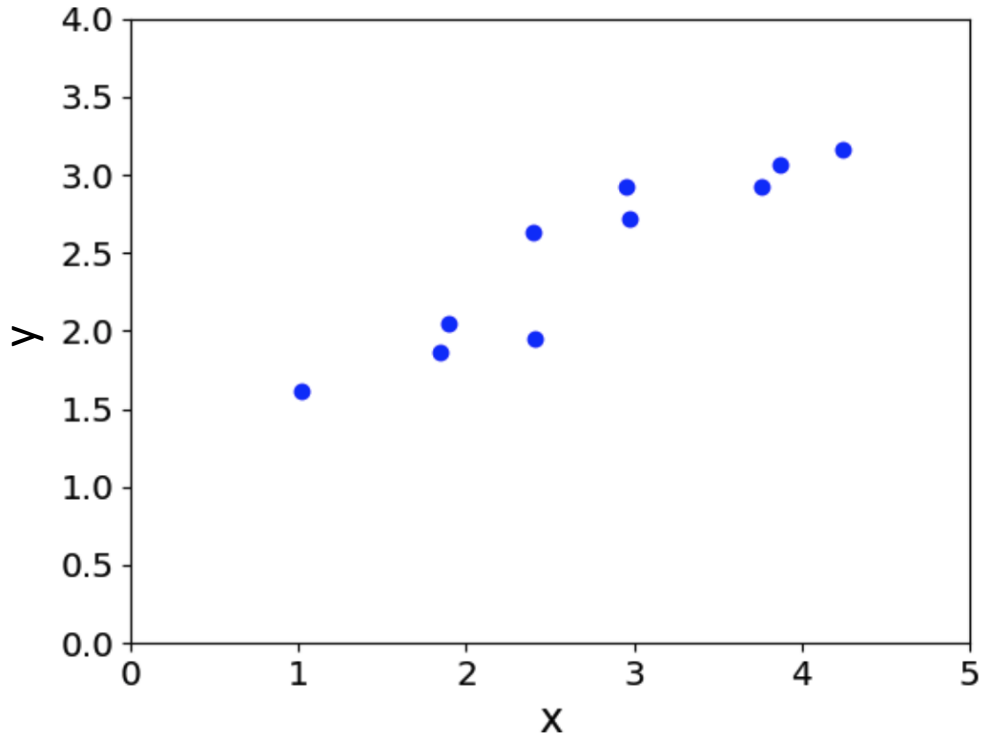


Image Credit: UofT CSC413 Lecture 1

Linear Regression

- Problem Specification (Scalar-Output Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

Linear Regression

- Problem Specification (Scalar-Output Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

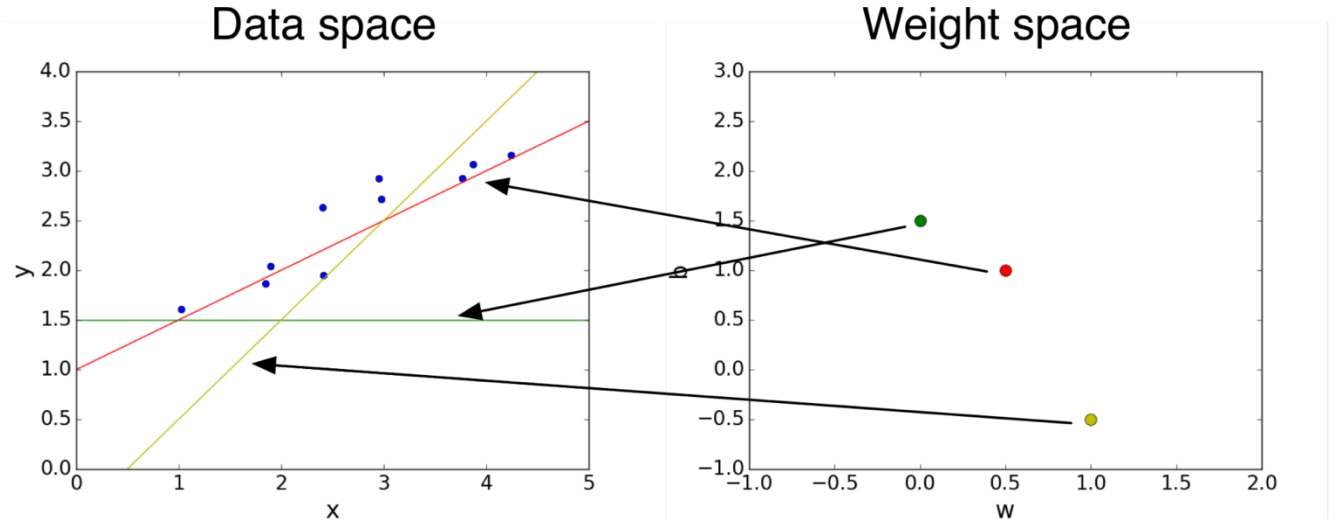


Image Credit: UofT CSC413 Lecture 1

Linear Regression

- Problem Specification (Scalar-Output Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

- Loss Design $L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$

Linear Regression

- Problem Specification (Scalar-Output Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

- Loss Design $L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$

1) Mean squared error (MSE), a.k.a., L2 loss $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$

Linear Regression

- Loss Design

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$$

1) MSE (L2 loss)

$$\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$$

2) L1 loss

$$\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_1$$

Linear Regression

- Loss Design

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N}\sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$$

1) MSE (L2 loss)

$$\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$$

2) L1 loss

$$\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_1$$

3) Smooth L1 loss

$$\ell(\hat{y}_n, y_n) = \begin{cases} \frac{1}{2\beta}\|\hat{y}_n - y_n\|_2^2, & \|\hat{y}_n - y_n\|_1 < \beta, \\ \|\hat{y}_n - y_n\|_1 - \frac{\beta}{2}, & \text{otherwise.} \end{cases} \quad (\beta > 0)$$

Quadratic near zero, linear for large errors.

A scaled version of the Huber loss. [3]

Shown: $\beta = 1$

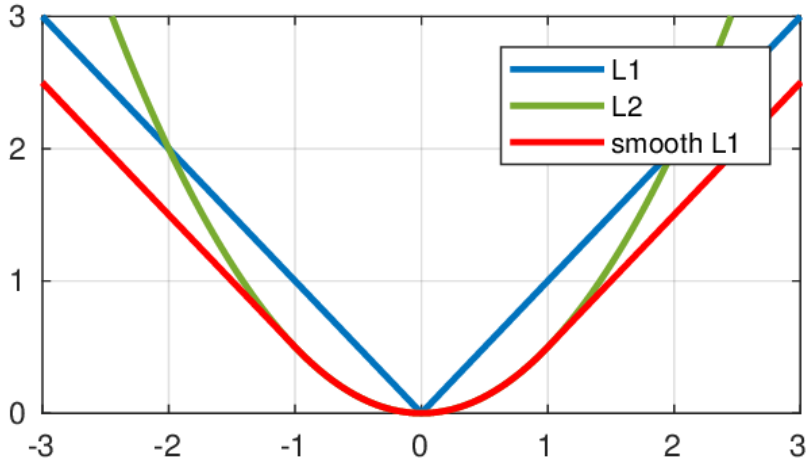


Figure: original lecture slides

Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - **Learning algorithm**
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

Linear Regression

- Learning Algorithm

We use half the MSE to simplify the derivatives.

$$L(w, b) = \frac{1}{2N} \sum_{n=1}^{N} (w^\top x_n + b - y_n)^2$$

The factor 1/2 preserves the minimizer and rescales the gradient.

Linear Regression

- Learning Algorithm

We use half the MSE to simplify the derivatives.

$$L(w, b) = \frac{1}{2N} \sum_{n=1}^{N} (w^\top x_n + b - y_n)^2$$

The factor 1/2 preserves the minimizer and rescales the gradient.

Let us write the prediction residual explicitly.

$$l_n = w^\top x_n + b - y_n, \qquad L(w, b) = \frac{1}{2N} \sum_{n=1}^{N} l_n^2$$

Linear Regression

- Learning Algorithm

The chain rule gives each component of the gradient.

$$l_n = w^\top x_n + b - y_n, \qquad L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} l_n^2$$

$$\frac{\partial L}{\partial w[i]} = \sum_{n=1}^{N} \frac{l_n}{N}\frac{\partial l_n}{\partial w[i]} = \frac{1}{N}\sum_{n=1}^{N} l_n x_n[i]$$

Collecting the components gives the weight gradient and the bias derivative.

$$\frac{\partial L}{\partial w} = \frac{1}{N}\sum_{n=1}^{N} l_n x_n, \qquad \frac{\partial L}{\partial b} = \frac{1}{N}\sum_{n=1}^{N} l_n$$

Each example contributes its prediction residual, weighted by its input.

Linear Regression

- One Gradient Descent Step

Consider one training example with one input feature.

$$x_1 = 2, \quad y_1 = 3, \quad w^{(0)} = 1, \quad b^{(0)} = 0, \quad \eta = 0.1$$

Compute the prediction, residual, loss, and both derivatives.

$$\hat{y}_1 = 2, \quad l_1 = -1, \quad L = \tfrac{1}{2}, \quad \frac{\partial L}{\partial w} = -2, \quad \frac{\partial L}{\partial b} = -1$$

Linear Regression

- One Gradient Descent Step

Consider one training example with one input feature.

$$x_1 = 2, \quad y_1 = 3, \quad w^{(0)} = 1, \quad b^{(0)} = 0, \quad \eta = 0.1$$

Compute the prediction, residual, loss, and both derivatives.

$$\hat{y}_1 = 2, \quad l_1 = -1, \quad L = \tfrac{1}{2}, \quad \frac{\partial L}{\partial w} = -2, \quad \frac{\partial L}{\partial b} = -1$$

Update both parameters simultaneously.

$$w^{(1)} = 1 - 0.1(-2) = 1.2, \qquad b^{(1)} = 0 - 0.1(-1) = 0.1$$

$$\hat{y}_1 = 1.2 \times 2 + 0.1 = 2.5, \qquad L = \tfrac{1}{2}(2.5 - 3)^2 = 0.125$$

Linear Regression

- Gradient Descent

Algorithm 1 GD Learning Algorithm

1: **Input**: GD step T, learning Rate η, initial (w^0, b^0)
2: **For** $t = 1, \cdots, T$
3: $\quad w^t = w^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial w}$
4: $\quad b^t = b^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial b}$
5: **Return** (w^T, b^T)

The learning rate controls the size of each step.

Linear Regression

- Gradient Descent

Algorithm 1 GD Learning Algorithm

1: **Input**: GD step T, learning Rate η, initial (w^0, b^0)
2: **For** $t = 1, \cdots, T$
3: $\quad w^t = w^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial w}$
4: $\quad b^t = b^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial b}$
5: **Return** (w^T, b^T)

The learning rate controls the size of each step.

Full batch: use all training examples. Minibatch SGD: use a random subset.

$$\frac{\partial L}{\partial w} \approx \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} l_n x_n$$

Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - Learning algorithm
- **Generalization and Model Evaluation**
  - **Validation, testing, and model selection**
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

Generalization and Model Evaluation

- Optimization and generalization

$f_{\mathcal{D}}$: predictor fitted to training dataset $\mathcal{D}$

Training loss $$L_{\text{train}}(f_{\mathcal{D}}) = \frac{1}{N}\sum_{n=1}^{N} \ell\big(f_{\mathcal{D}}(x_n), y_n\big)$$

Measures fit to the observed training examples.

Expected loss $$R(f_{\mathcal{D}}) = \mathbb{E}_{(x,y)\sim\mathbb{P}_{\text{data}}}\big[\ell\big(f_{\mathcal{D}}(x), y\big)\big]$$

Average over fresh examples with the fitted predictor held fixed.

Is the model at the last training step necessarily the best?

Generalization and Model Evaluation

- Validation and Testing

The goal is to generalize well to unseen data.

Generalization and Model Evaluation

- Validation and Testing

The goal is to generalize well to unseen data.

Training set: fit/learning model parameters.

Generalization and Model Evaluation

- Validation and Testing

The goal is to generalize well to unseen data.

Training set: fit/learning model parameters.

Validation set: tune hyperparameters and select a checkpoint.

Generalization and Model Evaluation

- Validation and Testing

The goal is to generalize well to unseen data.

Training set: fit/learning model parameters.

Validation set: tune hyperparameters and select a checkpoint.

Test set: evaluate the selected model without using the results to tune it.

Generalization and Model Evaluation

- Model selection in practice

Checkpoint selection

Save the checkpoint with the lowest validation loss.

Illustrative example

Checkpoint	**Validation loss**
Early	0.42
Middle	**0.31**
Final	0.48

k-fold cross-validation

With limited data, train on all but one fold and validate on that fold. Rotate the validation fold and average the scores. [2]

Keep the final test set separate from every model-selection decision.

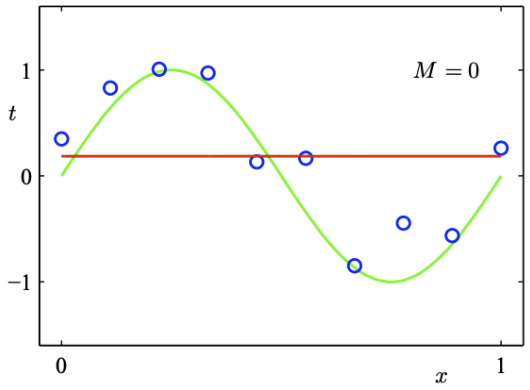
Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - **Underfitting and overfitting**
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

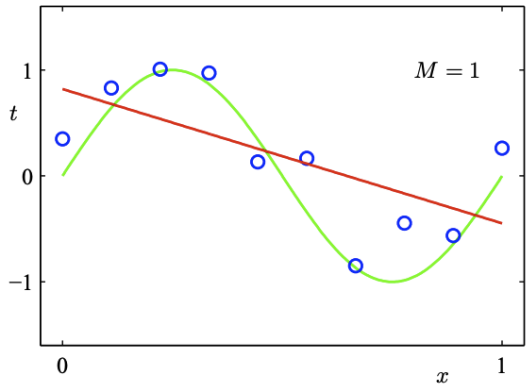
Generalization and Model Evaluation

- Underfitting and Overfitting

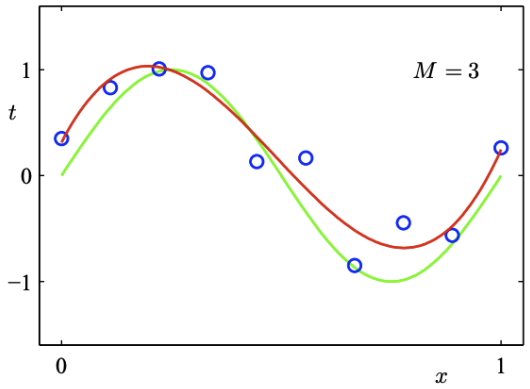
$y = w_0$



$y = w_0 + w_1 x$



$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$



$y = w_0 + w_1 x + \cdots + w_9 x^9$

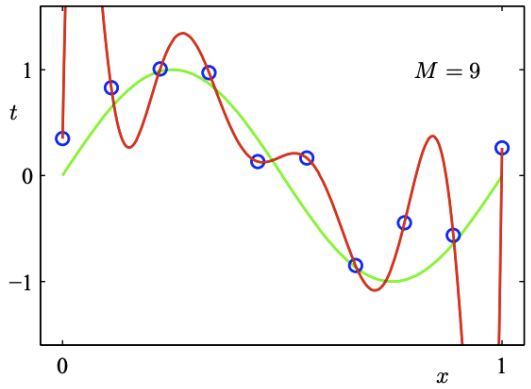
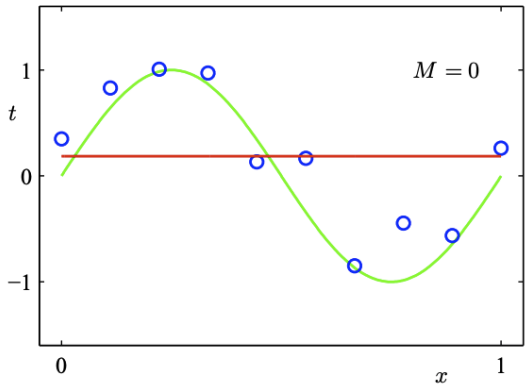


Image Credit: [2]

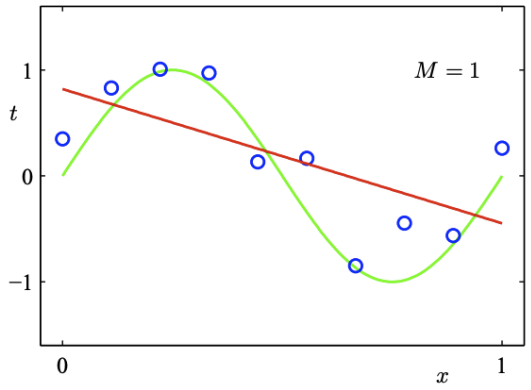
Generalization and Model Evaluation

- Underfitting and Overfitting

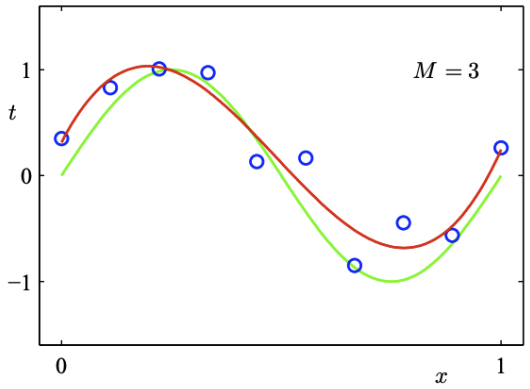
$y = w_0$



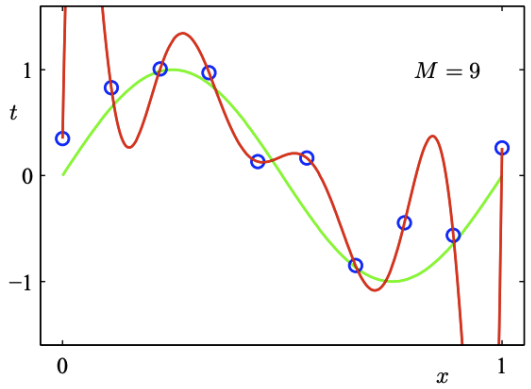
$y = w_0 + w_1 x$



$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$



$y = w_0 + w_1 x + \cdots + w_9 x^9$



Underfitting

The model misses the main pattern in the data.

Overfitting

The model fits training-specific details that do not generalize.

Image Credit: [2]

Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - **Bias–variance tradeoff**
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

Generalization and Model Evaluation

- Learning from different training sets

Fix one input and train repeatedly on datasets of the same size.

One dataset gives one prediction:

$$f_{\mathcal{D}}(x)$$

Average across training datasets:

$$\bar{f}(x) := \mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}}(x)]$$

Greater flexibility often lowers bias and raises variance.

Parameter count alone does not determine generalization.

Same data, different flexibility

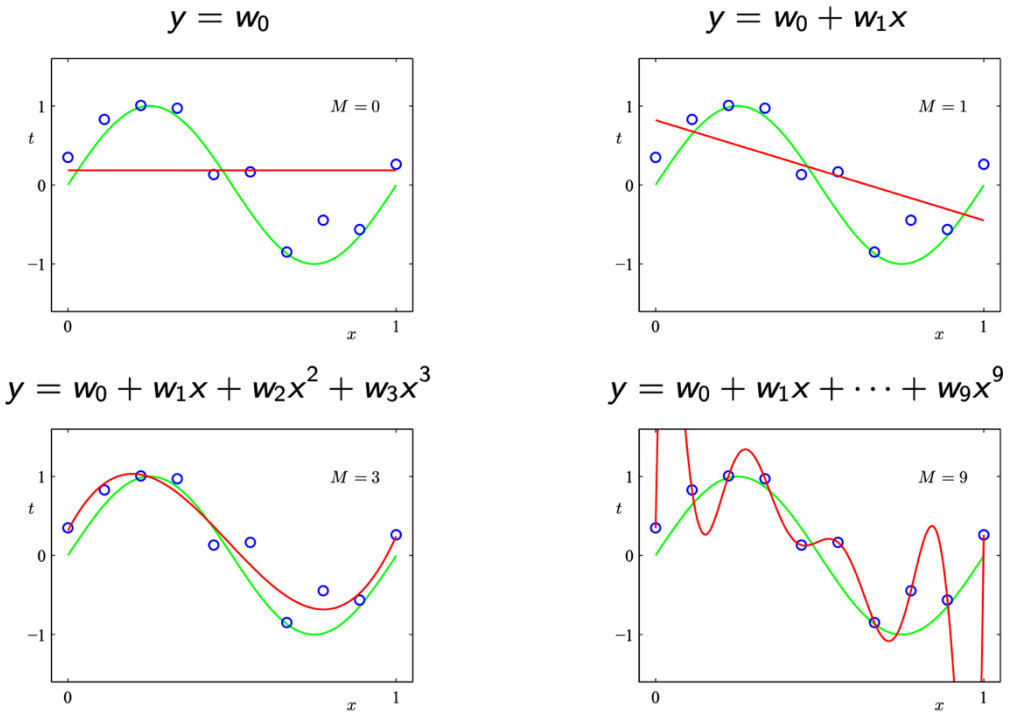


Image Credit: [2]

Generalization and Model Evaluation

- Bias, Variance, and Noise

At a fixed input: $\bar{y}(x) := \mathbb{E}[y \mid x], \qquad \bar{f}(x) := \mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}}(x)]$

Bias

Error in the average prediction.

$$\mathrm{Bias}(x) := \bar{f}(x) - \bar{y}(x)$$

Variance

Variation across training datasets.

$$\mathrm{Variance}(x) := \mathbb{E}_{\mathcal{D}}\left[(f_{\mathcal{D}}(x) - \bar{f}(x))^2\right]$$

Noise

Output variation unexplained by the input.

$$\mathrm{Noise}(x) := \mathbb{E}\left[(y - \bar{y}(x))^2 \mid x\right]$$

$$\mathbb{E}_{\mathcal{D},y}\left[(y - f_{\mathcal{D}}(x))^2 \mid x\right] = \mathrm{Bias}(x)^2 + \mathrm{Variance}(x) + \mathrm{Noise}(x)$$

Model size alone does not determine generalization; see Appendix B of the notes.

Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - **Logistic regression and binary cross-entropy**
  - Multiclass logistic regression and softmax
- Linear Models and Neural Networks

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

We can construct a threshold classifier (a discontinuous Heaviside step function) as

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

We can construct a threshold classifier (a discontinuous Heaviside step function) as

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The classification error rate (average 0-1 loss) is

$$\bar{\ell} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n \neq y_n\right]$$

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

We can construct a threshold classifier (a discontinuous Heaviside step function) as

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The classification error rate (average 0-1 loss) is

$$\bar{\ell} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n \neq y_n\right]$$

How can we perform gradient descent to learn the model?

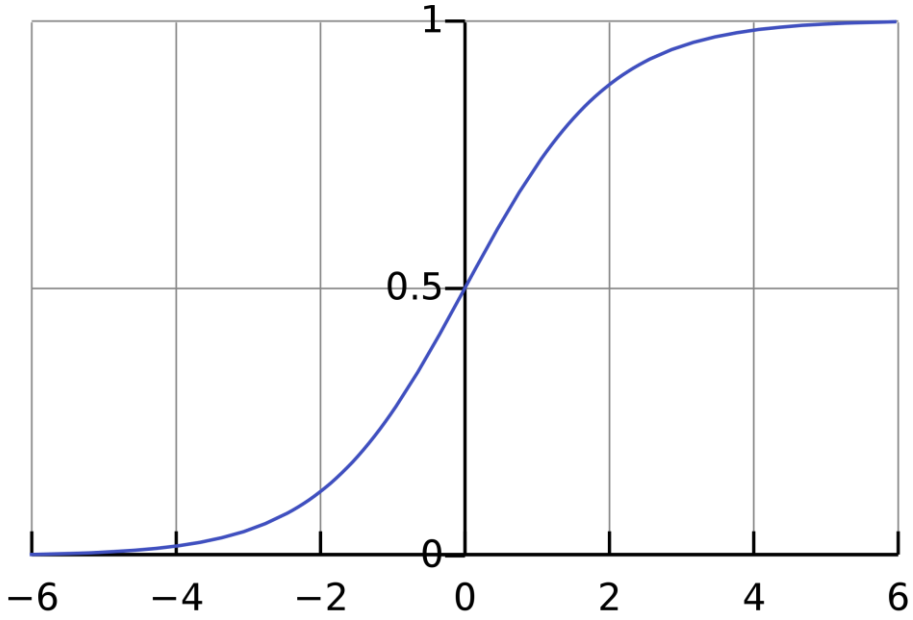
Linear Models for Classification

For the threshold classifier (a discontinuous Heaviside step function),

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

we use a logistic sigmoid to model the probability of class 1

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



Linear Models for Classification

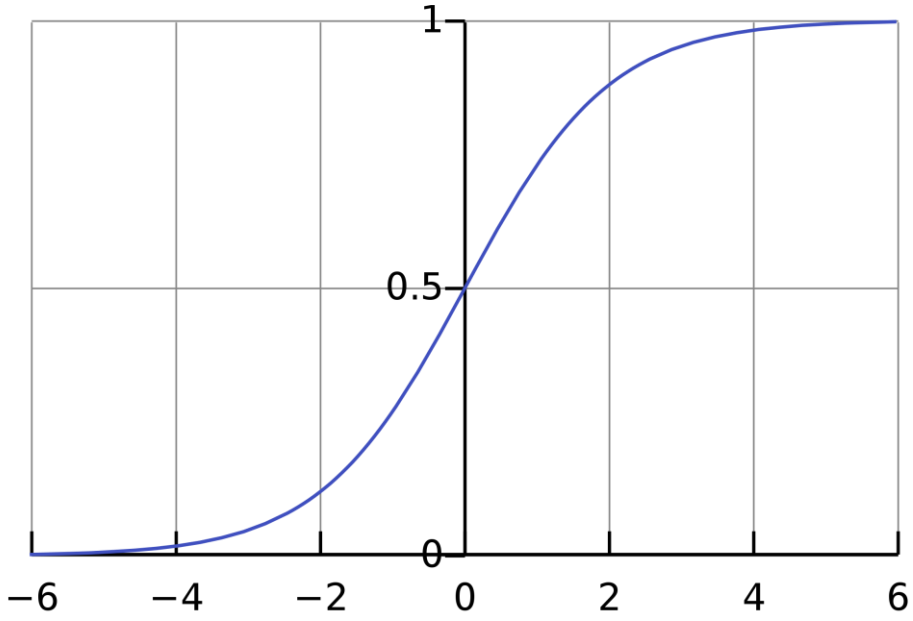
For the threshold classifier (a discontinuous Heaviside step function),

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

we use a logistic sigmoid to model the probability of class 1

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

$$\hat{y} = \sigma(w^\top x + b)$$



Linear Models for Classification

For the threshold classifier (a discontinuous Heaviside step function),

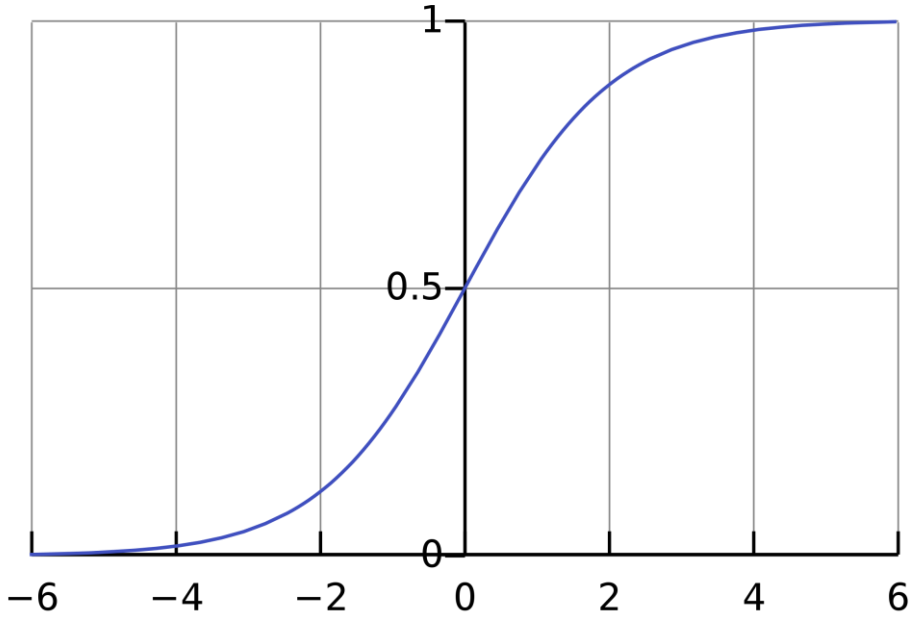
$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

we use a logistic sigmoid to model the probability of class 1

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

$$\hat{y} = \sigma(w^\top x + b)$$

This models the probability of class 1.



Linear Models for Classification

- Logistic Regression

$$\hat{y}_n = \sigma(f(x_n, w, b)), \qquad f(x_n, w, b) = w^\top x_n + b$$

A differentiable loss uses the probability assigned to the correct label.

$$\ell(\hat{y}_n, y_n) = -y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n)$$

Binary cross-entropy is the negative log-likelihood of a Bernoulli model. [2]

Linear Models for Classification

- Logistic Regression

$$\hat{y}_n = \sigma(f(x_n, w, b)), \qquad f(x_n, w, b) = w^\top x_n + b$$

A differentiable loss uses the probability assigned to the correct label.

$$\ell(\hat{y}_n, y_n) = -y_n \log \hat{y}_n - (1 - y_n) \log(1 - \hat{y}_n)$$

Binary cross-entropy is the negative log-likelihood of a Bernoulli model. [2]

For a positive example (y = 1), confidently wrong predictions receive a larger loss.

$\hat{y}_n$	0.1	0.6	0.9
$\ell(\hat{y}_n, 1) = -\log \hat{y}_n$	2.303	0.511	0.105

Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - **Multiclass logistic regression and softmax**
- Linear Models and Neural Networks

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \ldots, K\}$$

$$f(x, W, b) = Wx + b, \qquad W \in \mathbb{R}^{K \times D}, \quad b \in \mathbb{R}^{K \times 1}$$

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \ldots, K\}$$

$$f(x, W, b) = Wx + b, \qquad W \in \mathbb{R}^{K \times D}, \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]}_{\text{k-th entry is 1}}$$

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \ldots, K\}$$

$$f(x, W, b) = Wx + b, \qquad W \in \mathbb{R}^{K \times D}, \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]}_{\text{k-th entry is 1}}$$

By doing so, we can conveniently use the cross-entropy as the loss.

For mutually exclusive classes, the probabilities must sum to 1.

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \ldots, K\}$$

$$f(x, W, b) = Wx + b, \qquad W \in \mathbb{R}^{K \times D}, \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]}_{\text{k-th entry is 1}}$$

By doing so, we can conveniently use the cross-entropy as the loss.

For mutually exclusive classes, the probabilities must sum to 1.

Instead, we can use the softmax function, which outputs a valid probability distribution of a categorical RV with K states,

$$\text{softmax}(x)[i] = \frac{\exp(x[i])}{\sum_{k=1}^{K} \exp(x[k])}$$

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n)|n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \ldots, K\}$$

$$f(x, W, b) = Wx + b, \qquad W \in \mathbb{R}^{K \times D}, \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]}_{\text{k-th entry is 1}}$$

By doing so, we can conveniently use the cross-entropy as the loss.

For mutually exclusive classes, the probabilities must sum to 1.

Instead, we can use the softmax function, which outputs a valid probability distribution of a categorical RV with K states,

$$\text{softmax}(x)[i] = \frac{\exp(x[i])}{\sum_{k=1}^{K} \exp(x[k])}$$

It is called *logits* and the whole model is called *multiclass logistic regression*

Linear Models for Classification

- Multiclass Cross-Entropy

For a one-hot target, only the correct class contributes to the loss.

$$\ell(\hat{y}_n, y_n) = -\sum_{k=1}^{K} \mathbf{1}[y_n = k] \log \hat{y}_n[k] = -\log \hat{y}_n[y_n]$$

A larger probability for the correct class gives a smaller loss.

Linear Models for Classification

- Multiclass Cross-Entropy

For a one-hot target, only the correct class contributes to the loss.

$$\ell(\hat{y}_n, y_n) = -\sum_{k=1}^{K} \mathbf{1}[y_n = k] \log \hat{y}_n[k] = -\log \hat{y}_n[y_n]$$

Example: three classes, with class 2 as the correct label.

$$K = 3, \quad y_n = 2, \quad f(x_n, W, b) = \begin{bmatrix} 0 \\ \log 2 \\ 0 \end{bmatrix} \qquad \hat{y}_n = \begin{bmatrix} 1/4 \\ 1/2 \\ 1/4 \end{bmatrix}, \qquad \ell(\hat{y}_n, 2) = -\log(1/2) = \log 2 \approx 0.693$$

Linear Models for Classification

- Implementation

Use a combined loss from torch.nn.functional (F) that accepts logits. [3]

Binary classification

F.binary_cross_entropy_with_logits(logits, targets)

Multiclass classification

F.cross_entropy(logits, class_indices)

These losses handle sigmoid or log-softmax internally for numerical stability.

Outline

- Statistical Learning Setup
- Linear Regression
  - Model and loss
  - Learning algorithm
- Generalization and Model Evaluation
  - Validation, testing, and model selection
  - Underfitting and overfitting
  - Bias–variance tradeoff
- Linear Classification
  - Logistic regression and binary cross-entropy
  - Multiclass logistic regression and softmax
- **Linear Models and Neural Networks**

Linear Models and Neural Networks

- What Changes in an MLP?

A linear model uses the given input features.

An MLP learns a nonlinear feature representation.

$$\underbrace{Wx + b}_{\text{linear classifier}} \longrightarrow \underbrace{Wh(x) + b}_{\text{classifier with learned features}}$$

The prediction loss, gradient updates, and validation procedure remain familiar.

Can a linear classifier separate every pattern in the original input space?

References

[1] Vapnik, V. (1999). The Nature of Statistical Learning Theory. Springer.
[2] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
[3] PyTorch Documentation. CrossEntropyLoss, BCEWithLogitsLoss, and SmoothL1Loss.

Questions?