Deep Learning

Probability and Math Basics

Logistics

- Homework 1 is out.
  - It covers mostly math/machine learning review material
  - It is long! Start now!
  - Turn it in via gradescope – instructions on the HW
- Piazza is operational
- Today:
  - Review of concepts/definitions from probability and machine learning.

Probability and Statistics

- Probability models our empirical observations of "random" events.
- The theory makes testable predictions
- Where do we use probability and statistics in deep learning?

Expected Value

- *Expected Value, Expectation,* and *mean* all mean the same thing:

$$E[V] = \int_X x\, p(x) dx$$

- The *law of large numbers* states that if $V_1, V_2, \ldots, V_n$ are independent and identically distributed random variables with expected value $E[V]$, then we have

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} V_i = E[V]$$

Variance and Covariance

- The *variance* of a random variable is a measure of how likely the value is to be close to the mean.

$$Var(V) = E[(V - E[V])^2]$$

- Often, $Var(V)$ is represented by σ^2 and $E[V]$ is represented by μ.
- *Covariance* is a property of two random variables:
- $Cov(V, W) = E[(V - E[V])(W - E[W])]$

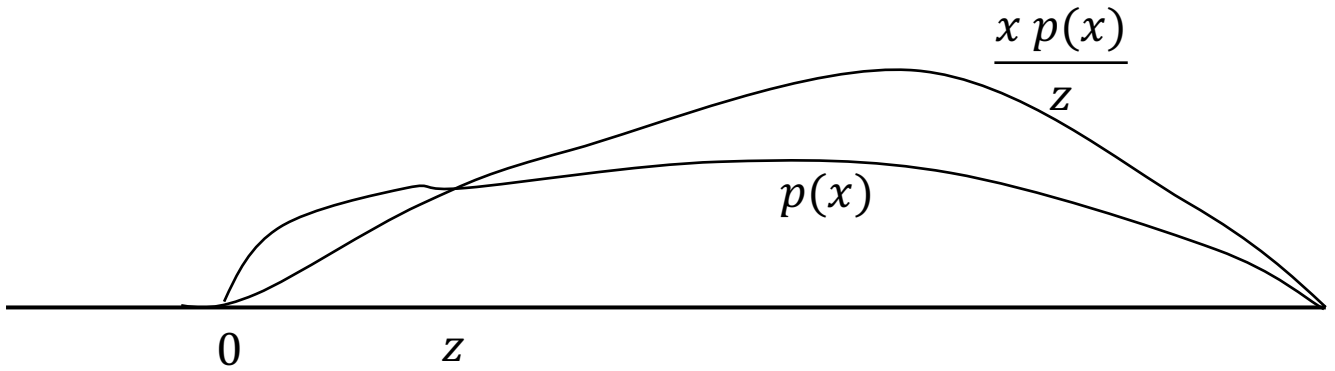
Properties of Expectation and Variance

- $E[V + W] = E[V] + E[W]$
- $E[cV] = cE[V]$
- $Var(V + W) = Var(V) + Var(W) + 2Cov(V, W)$
- $Var(cV) = c^2 Var(V)$
- Markov's inequality: If V is guaranteed to be non-negative, then

$$P[V > z] \leq \frac{E[V]}{z}$$

Proof of Markov Inequality

$$P[V > z] = \int_z^\infty p(x)dx$$
$$\leq \frac{1}{z}\int_z^\infty x\, p(x)dx$$
$$\leq \frac{1}{z}\int_0^\infty xp(x)dx = \frac{E[V]}{z}$$



Markov Inequality and Variance

- $(V - E[V])^2$ is always positive. Markov inequality then says that for any z,

$$P[(V - E[V])^2 > z] \leq \frac{Var(V)}{z}$$

Markov's inequality is possibly *the most important* probability fact used in machine learning.

Union Bound

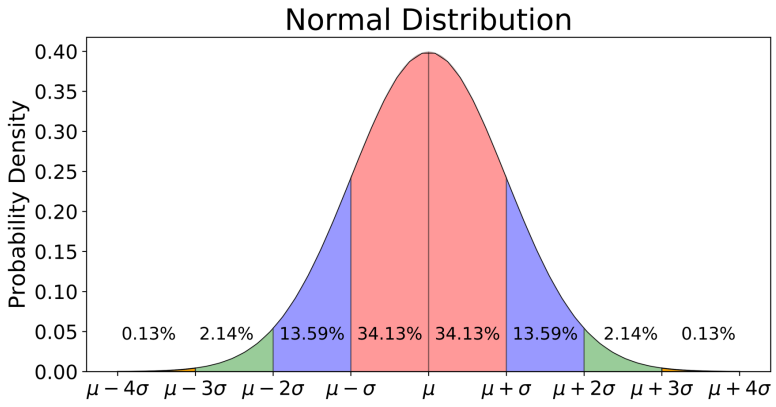
- Remember that $P[A\ OR\ B] \leq P[A] + P[B]$.
- Suppose $A_1, \ldots, A_N$ are all "bad" events, each of which has probability at most δ of happening.
- Can we bound the probability that NO bad events happen?

$$P[None\ of\ A_i] \geq 1 - N\delta$$

Common Densities

- Gaussian/Normal Distribution $N(\mu, \sigma^2)$:

$$p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$



https://towardsdatascience.com/understanding-the-68-95-99-7-rule-for-a-normal-distribution-b7b7cbf760c2

Common Densities Cont.

- Multivariate Gaussian:

$$p(x;\mu,\Sigma) \propto \frac{1}{\sqrt{\det(\Sigma)}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

  - μ is a vector and Σ is a symmetric positive-definite *covariance matrix*.
  - Linear combination of the Gaussian variables has a Gaussian distribution (what is the mean and variance?).
- Poisson Distribution: distribution over positive integers, parameterized by λ.

$$p(x;\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Whiteboard

$$p(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

$$p(x;\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Central Limit Theorem

- The normal distribution is particularly ubiquitous because of the *central limit theorem*.
- If $V_1, \ldots, V_N$ are independent (but not necessarily identically distributed) random variables, and $S = \frac{1}{N}(V_1 + \cdots + V_N)$, then S is approximately $N(\mu, \sigma^2)$ for $\mu = E[S]$ and $\sigma^2 = Var(\mathrm{S})$.

Machine Learning

- Machine learning is best for solving problems when:

1. You do not know how to write down the answer.
   - You should probably not use machine learning to solve quadratic equations or compute the rank of a matrix.

Visually Identifying Rank

David F. Fouhey, *Mathematicians Hate Him!*
Daniel Maturana, *Random Forester* Rufus von Woofles, *Good Boy*

Abstract—The visual estimation of the rank of a matrix has eluded researchers across a myriad of disciplines many years. In this paper, we demonstrate the successful visual estimation of a matrix's rank by treating it as a classification problem. When tested on a dataset of tens-of-thousands of colormapped matrices of varying ranks, we not only achieve state-of-the-art performance, but also distressingly high performance on an absolute basis.

Index Terms—perceptual organization; vitamin and rank deficiencies; egalitarianism in the positive-semi-definite cone; PAC bounds for SVDs; class-conscious norms

Typical Machine Learning Flow

1. Gather a "training dataset" of inputs/outputs
$$S_{train} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$
2. Choose a *hypothesis class* H. This is just a set of functions $f: X \to Y$.
3. Using the training data and some "training algorithm", identify some function $\hat{f} \in H$ such that ideally $\hat{f}(x) \approx y$.
4. Test $\hat{f}$ on a new "testing dataset" S_{test}.

Supervised Learning

- We want to build a "machine" or "program" that takes inputs x and produces output $\hat{y}$ such that $\hat{y} \approx y$ for some "true value" y

The Learning Problem

Digit Recognition

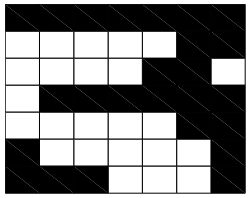


Image Classification

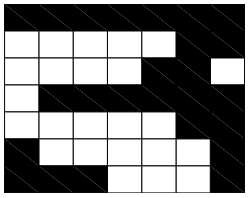
- Steps
  - —entertain a (biased) set of possibilities (hypothesis class)
  - —adjust predictions based on available examples (estimation)
  - —rethink the set of possibilities (model selection)
- Principles of learning are "universal"
  - —society (e.g., scientific community)
  - —animal (e.g., human)
  - —machine

Slide credit: Aykut Erdem // Koç University // Fall 2022

Hypothesis class

- Representation: examples are binary vectors of length $d = 64$

$$\mathbf{x} = [111\ldots0001]^T \; =$$



and labels $y \in \{-1, 1\}$ ("no","yes")

- The mapping from examples to labels is a "**linear classifier**"

$$\hat{y} = \text{sign}\left(\theta \cdot \mathbf{x}\right) = \text{sign}\left(\theta_1 x_1 + \ldots + \theta_d x_d\right)$$

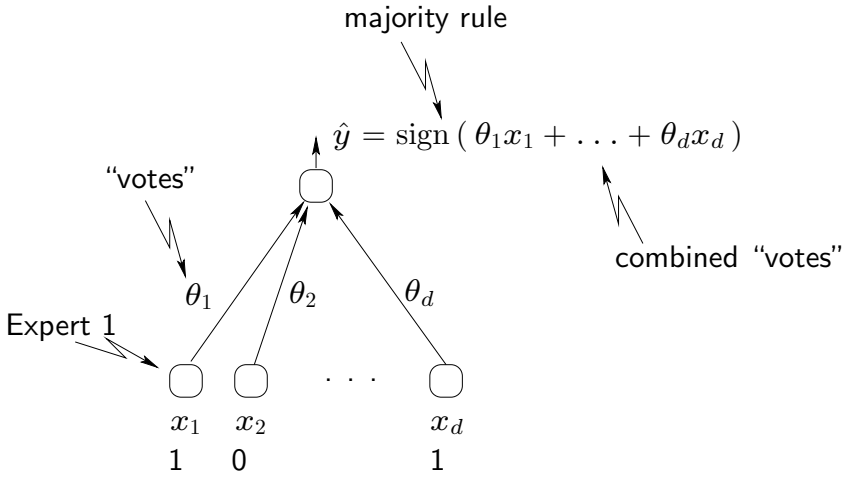
where θ is a vector of **parameters** we have to learn from examples.

Linear classifier/experts

- We can understand the simple linear classifier

$$\hat{y} = \text{sign}\left(\theta \cdot \mathbf{x}\right) = \text{sign}\left(\theta_1 x_1 + \ldots + \theta_d x_d\right)$$

as a way of combining expert opinion (in this case simple binary features)



Estimation

$\mathbf{x}$	y
011111100111001000000010000000100111111011101111100111011111 0001	+1
000111110000001100000111000001100111111011111100111111110000 0011	+1
111111100000011000001100011111100000011110000011111000110111 1111	−1
...	...

- How do we adjust the parameters θ based on the labeled examples?

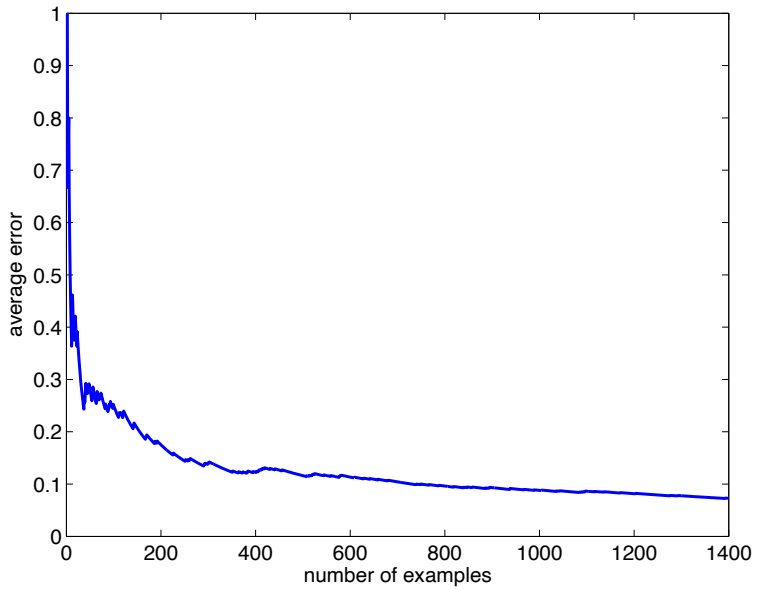
$$\hat{y} = \text{sign}\left(\theta \cdot \mathbf{x}\right)$$

For example, we can simply refine/update the parameters whenever we make a mistake (**perceptron algorithm**):

$$\theta_i \leftarrow \theta_i + y\, x_i,\ i = 1, \ldots, d \quad \text{if prediction was wrong}$$

Evaluation

- Does the simple mistake driven algorithm work?



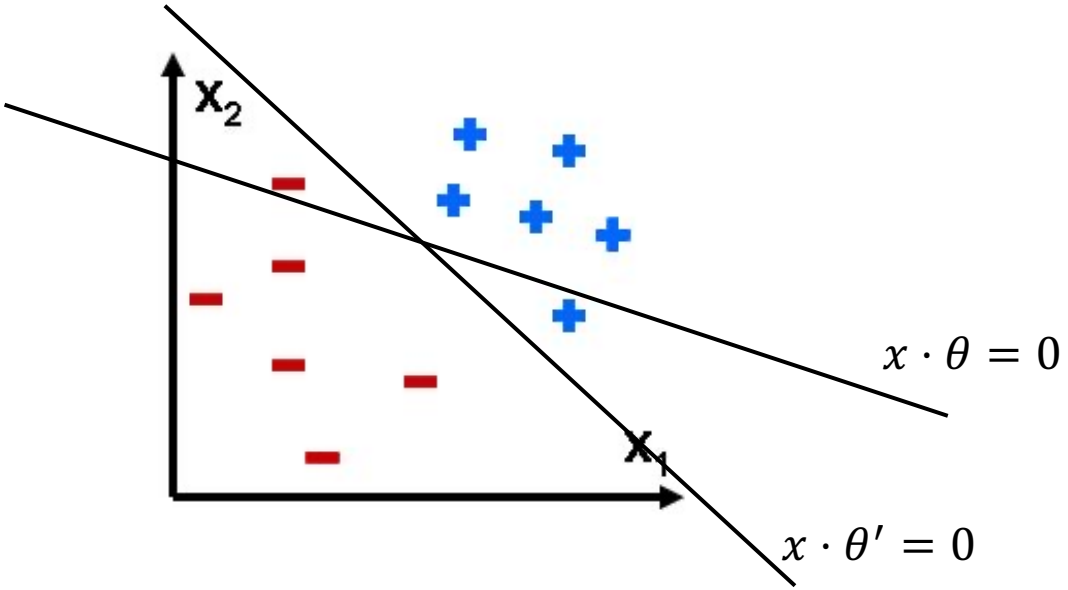
(average classification error as a function of the number of examples and labels seen so far)

Parametrized Hypothesis Class

- Instead of an abstract "set of functions", think of a *single* function $f: X \times \Theta \to Y$. We define H by
$$H = \{ x \mapsto f(x, \theta) \mid \theta \in \Theta \}$$
- Finding $\hat{f} \in H$ now means finding $\hat{\theta} \in \Theta$ and setting $\hat{f}(x) = f(x, \hat{\theta})$.
- Θ is called the parameter space, $\hat{\theta}$ is called the parameter estimate, $f(\mathrm{x}, \theta)$ is the model.

Example: Linear Classification

- Suppose $x \in \mathbb{R}^2$ and $y \in \{\pm 1\}$ (e.g. a binary classification task).



- $\Theta = \mathbb{R}^2, f(x, \theta) = sign(x \cdot \theta)$

Linear classifier: image classification

[32x32x3]
array of numbers 0...1
(3072 numbers total)

image parameters

$$f(x, W)$$

10 numbers, indicating class scores

Linear classifier: image classification

$$f(x, W) = Wx$$

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

Linear classifier: image classification

$$f(x, W) = Wx$$

3072x1

10x1

10x3072

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"

Linear classifier: image classification

$$f(x, W) = Wx$$

3072x1

(+b) 10x1

10x1

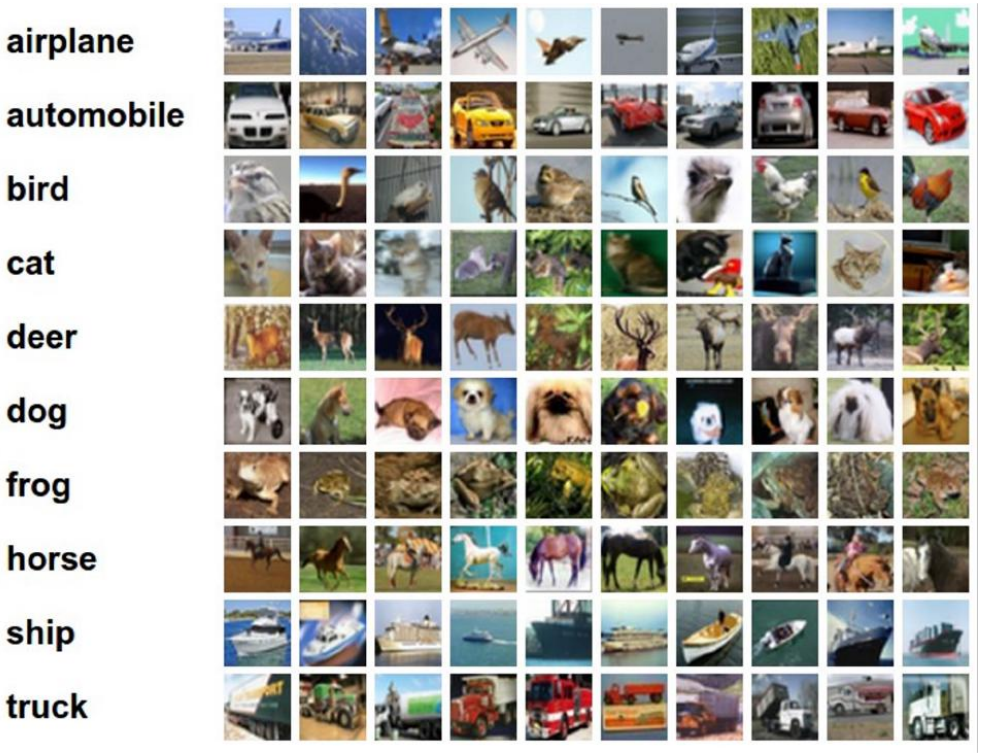
10x3072

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"

Interpreting a Linear Classifier

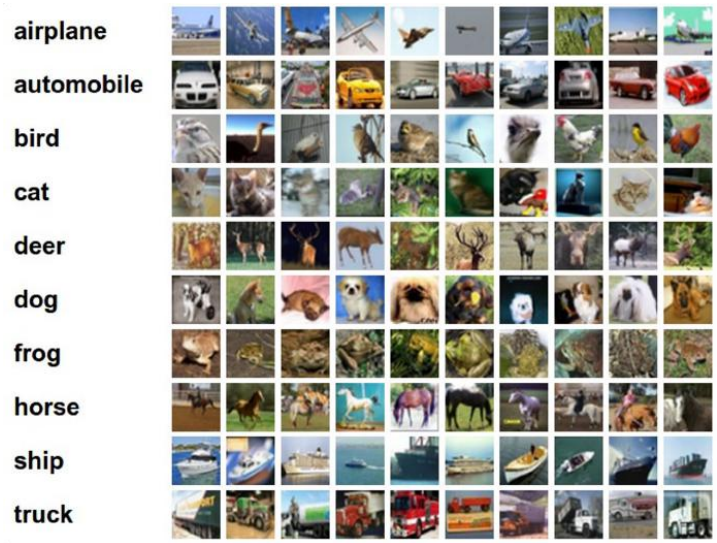


$$f(x_i, W, b) = Wx_i + b$$

Q: what does the linear classifier do, in English?

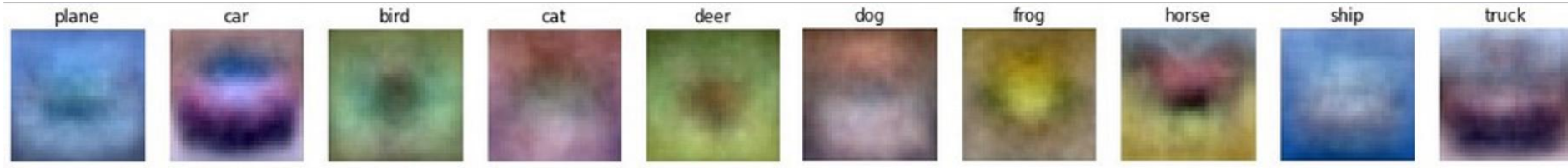
Slide credit: Aykut Erdem // Koç University // Fall 2022

Interpreting a Linear Classifier

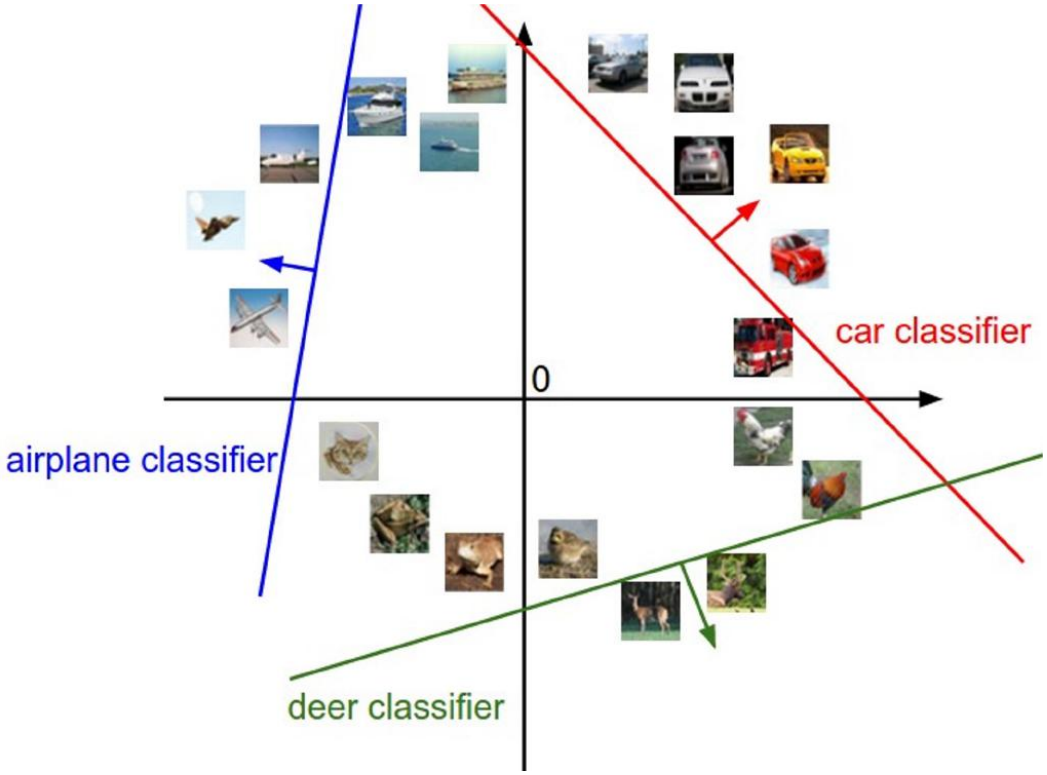


$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:



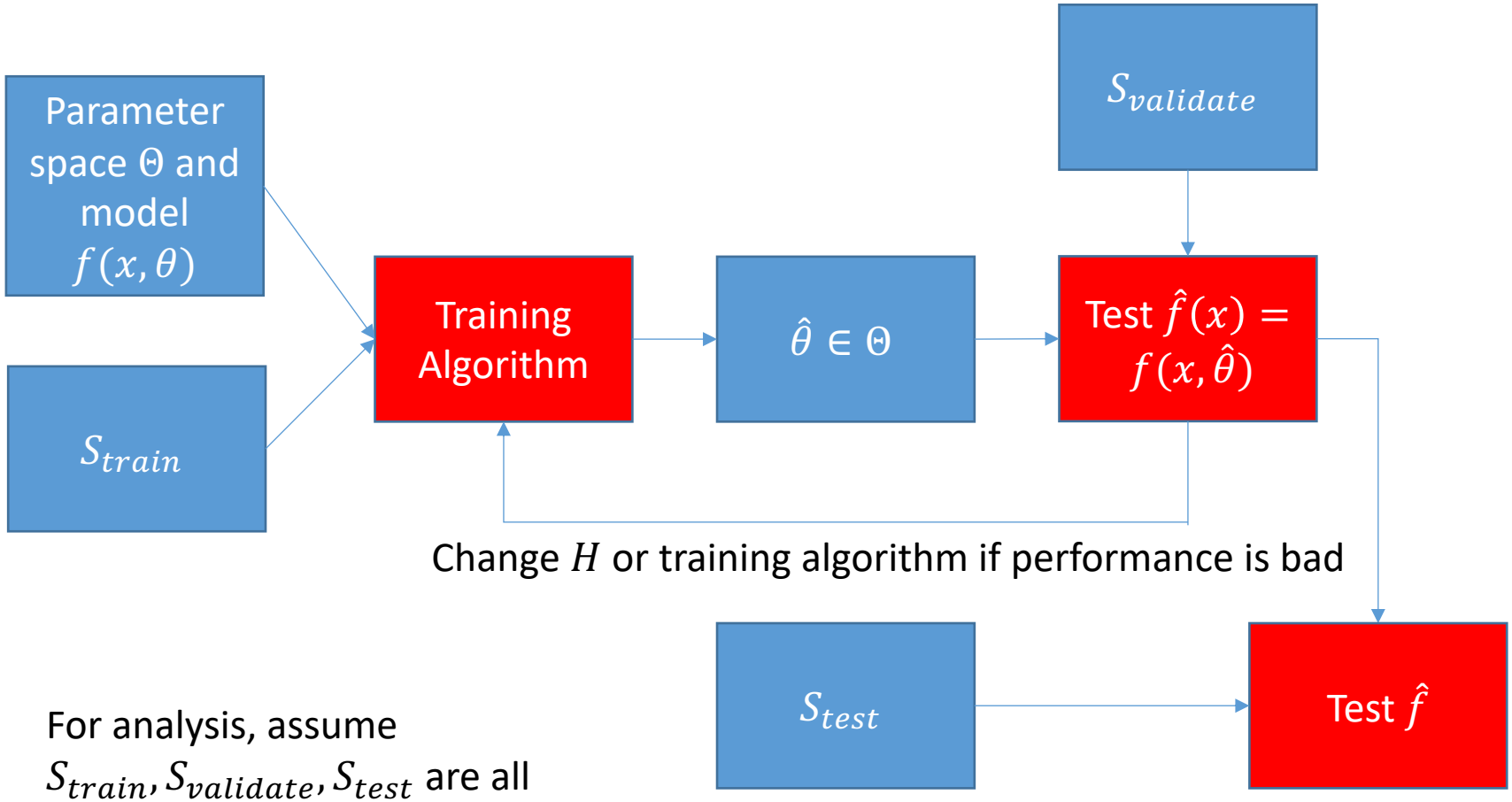
Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

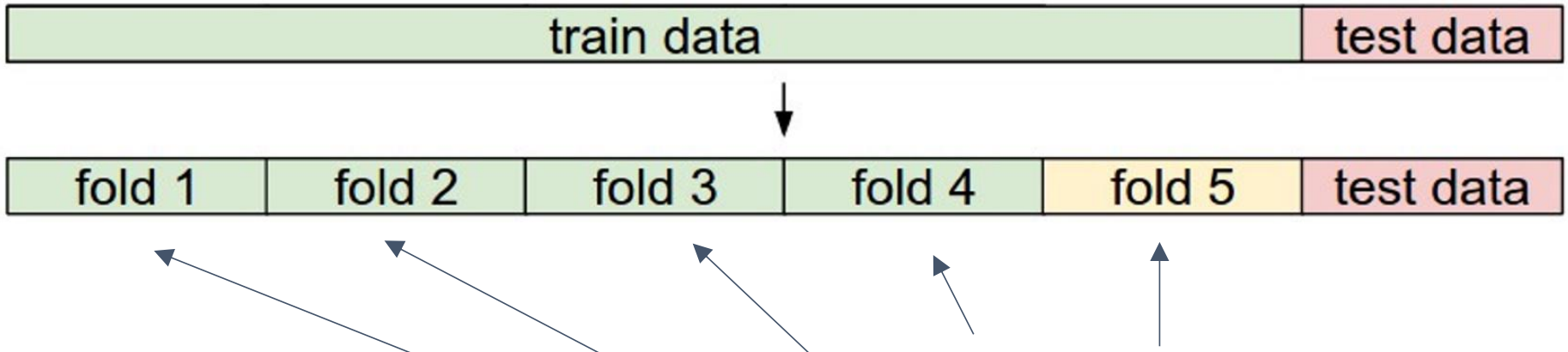
[32x32x3]
array of numbers 0...1
(3072 numbers total)

Supervised Learning Flowchart



For analysis, assume $S_{train}, S_{validate}, S_{test}$ are all i.i.d.

Technically, should only use testing set once.
Unfortunately, In practice, often $S_{test} = S_{validate}$.



Cross-validation

cycle through the choice of which fold is the validation fold, average results.

I.I.D. Training/Testing Data

- The vast majority of machine learning algorithms and analysis assume that each (x, y) pair in both S_{train} and S_{test} are sampled **independently** from the same distribution.
- This is assumption is frequently invalid, but seems to be a "good enough" model for many settings.
- This course (Deep Learning), is mostly about *extremely empirically successful ways to choose the model* $f(x, \theta)$.

Examples of cost functions

- What is a natural cost function for a classification task?

$$\ell(x, y, \theta) = 0 \text{ if } f(x, \theta) = y, 1 \text{ otherwise.}$$

  - $L_S(\theta)$ is the average number of mistakes in S

- What about for regression tasks?

$$\ell(x, y, \theta) = ||f(x, \theta) - y||^2$$

loss function

quantifies agreement between the predicted scores and ground truth labels

$$L_i = \max(0, 1 - y_i[\mathbf{w} \cdot \mathbf{x}_i + b])$$

hinge loss function

scoring function

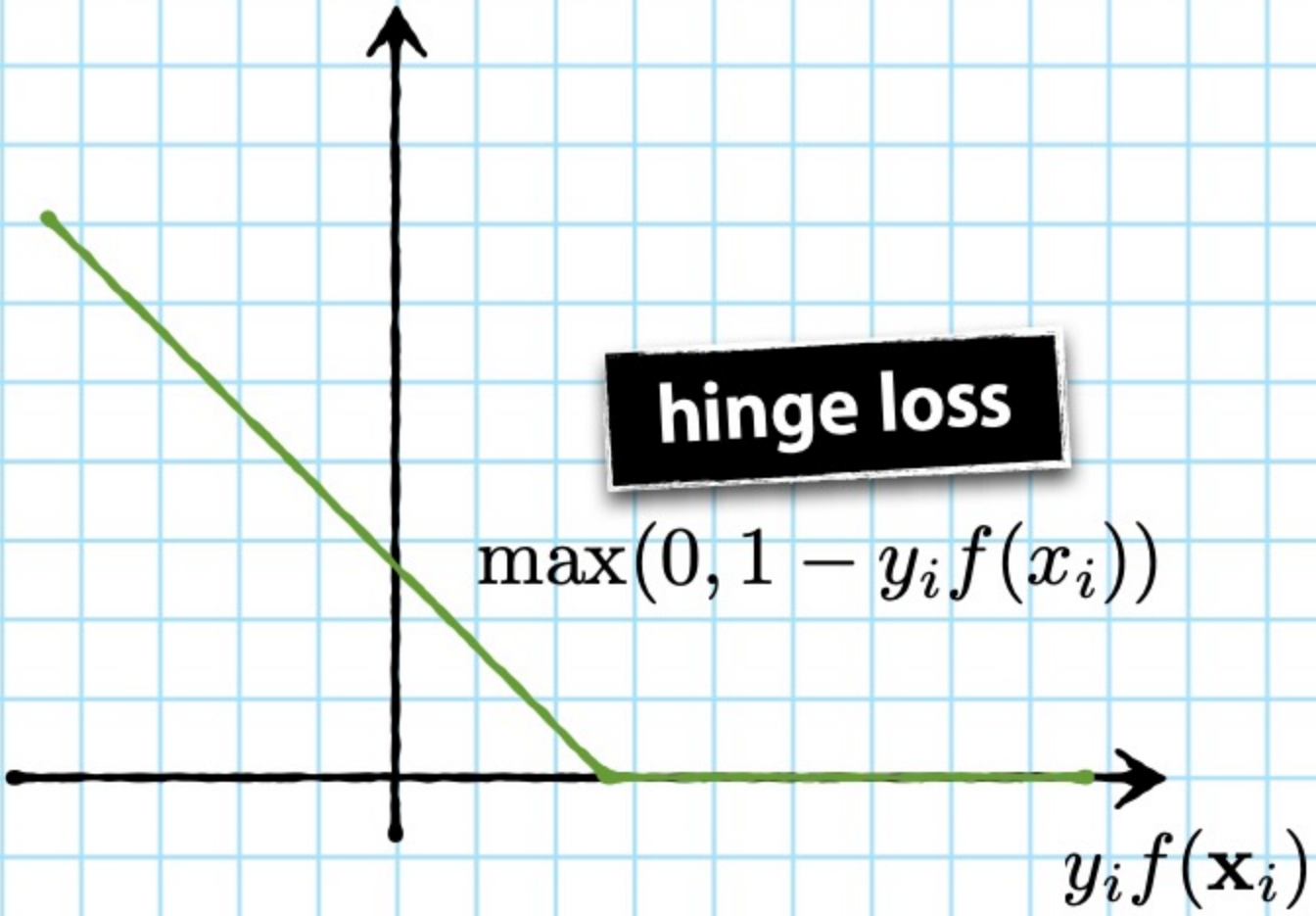
$$L_i = \max(0, 1 - y_i[\mathbf{w} \cdot \mathbf{x}_i + b])$$

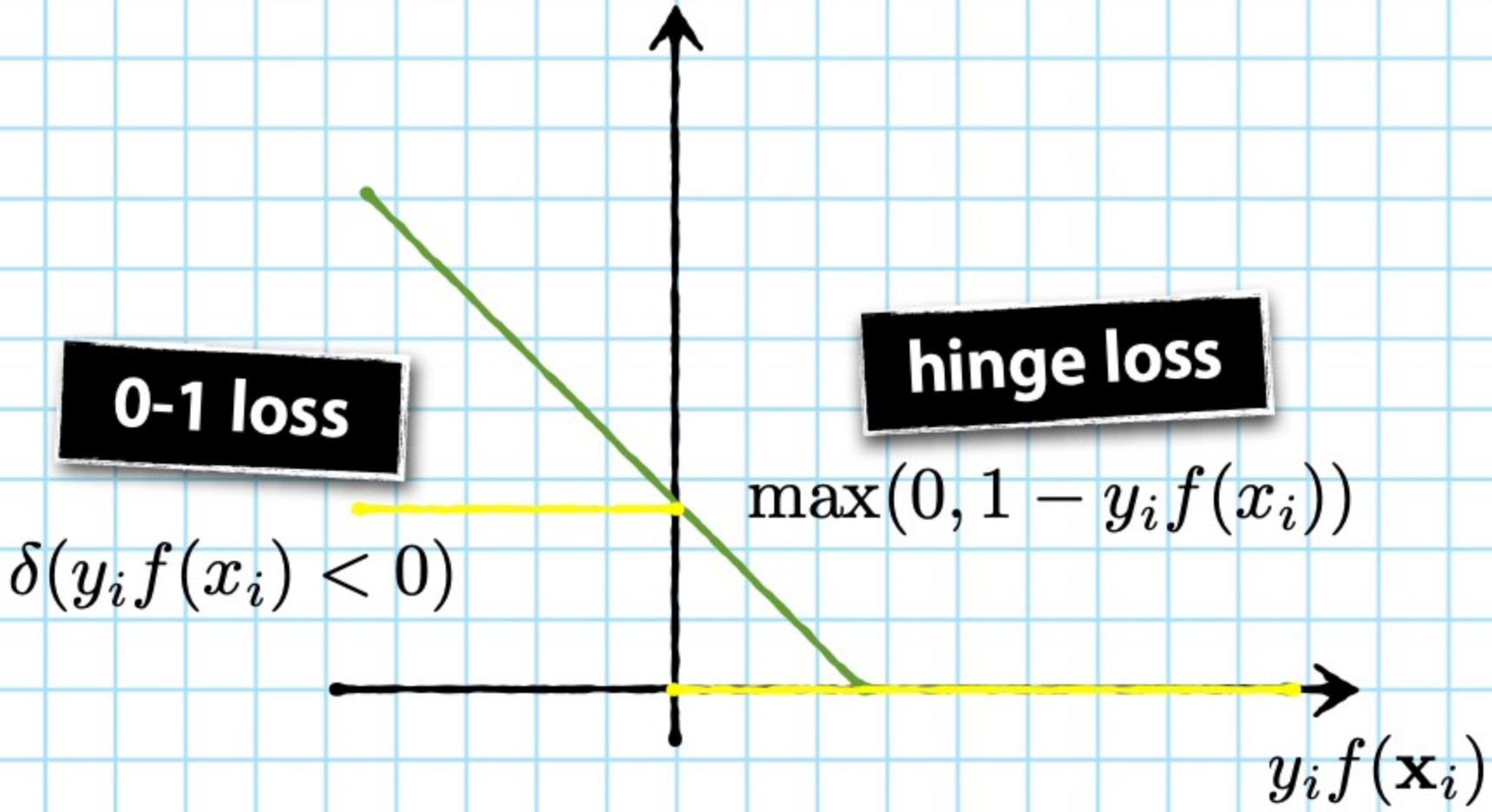
$$L_i = \max(0, 1 - y_i[\mathbf{w} \cdot \mathbf{x}_i + b])$$

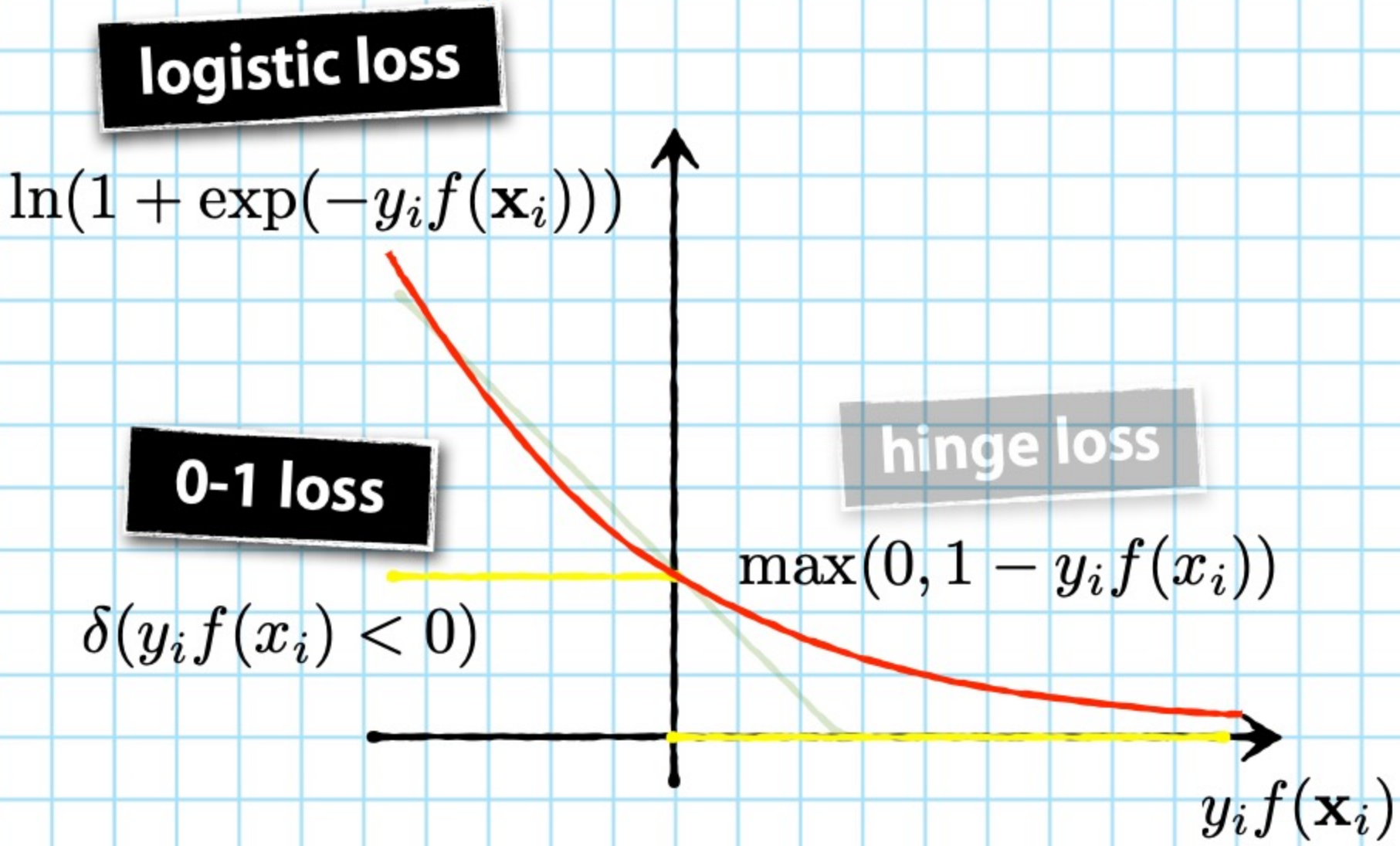
training set label, 1 or -1

loss margin

$$L_i = \max(0, 1 - y_i[\mathbf{w} \cdot \mathbf{x}_i + b])$$


hinge loss
$\max(0, 1 - y_i f(x_i))$
$y_i f(\mathbf{x}_i)$


0-1 loss
hinge loss
$\max(0, 1 - y_i f(x_i))$
$\delta(y_i f(x_i) < 0)$
$y_i f(\mathbf{x}_i)$


logistic loss
$\ln(1 + \exp(-y_i f(\mathbf{x}_i)))$
0-1 loss
$\delta(y_i f(x_i) < 0)$
hinge loss
$\max(0, 1 - y_i f(x_i))$
$y_i f(\mathbf{x}_i)$

Statistical regression models

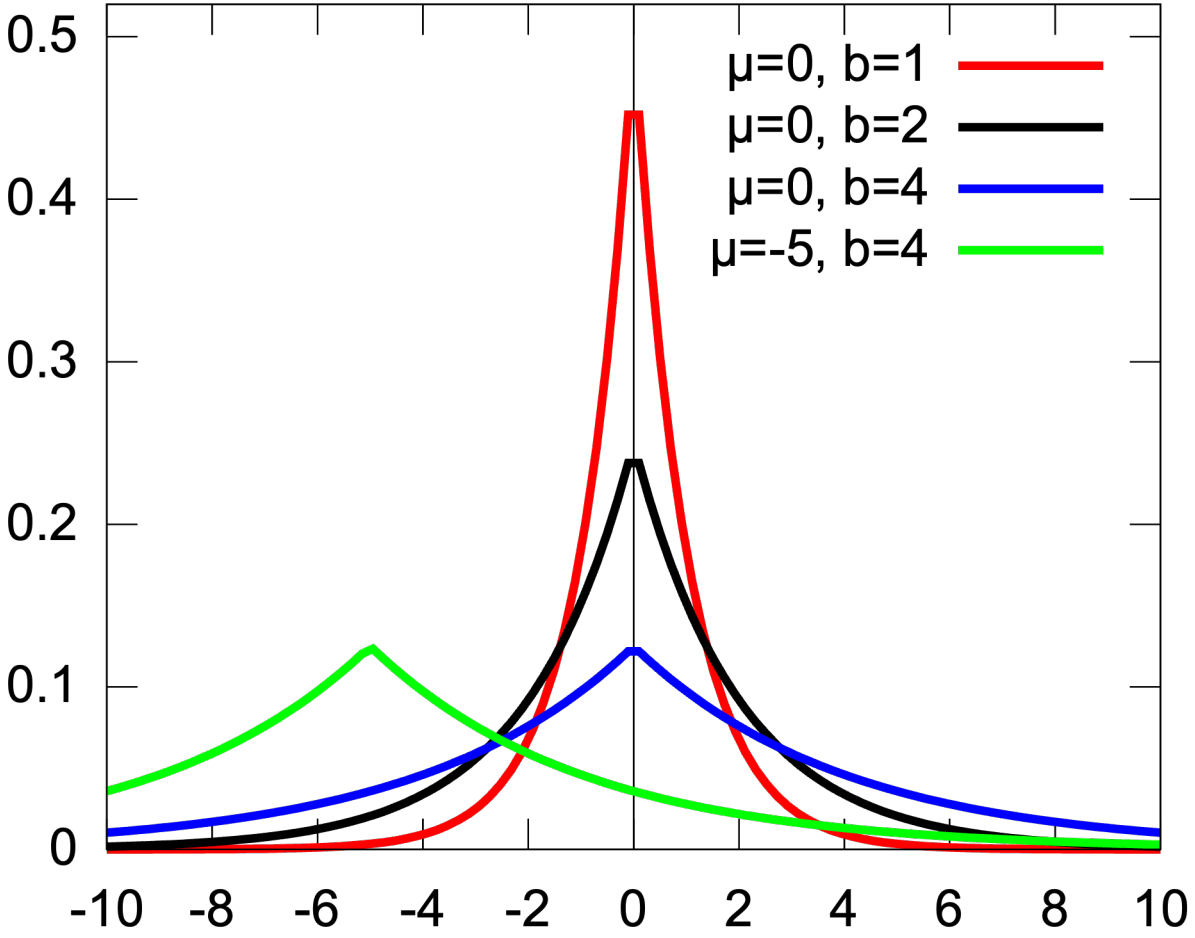
— model formulation, motivation
— maximum likelihood estimation

Maximum Likelihood Estimator

- The MLE is a training algorithm used when the hypothesis class H is a set of probability distributions:

$$H = \{ p(y|x;\theta) \mid \theta \in \Theta \}$$

Maximum Likelihood Example

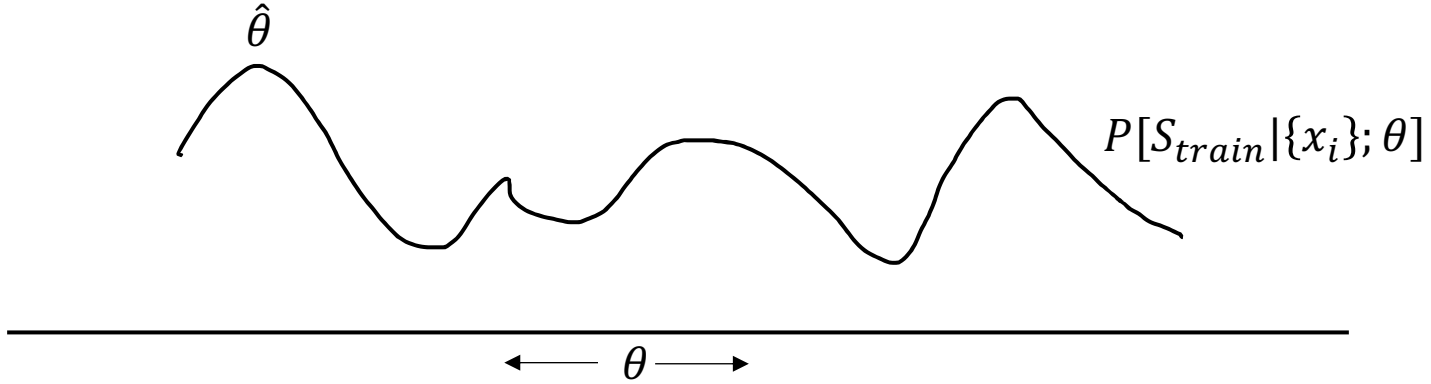


Likelihood Function

- Assuming each data point (x, y) is i.i.d. according to $p(y|x;\theta)$ for some θ, we have:

$$P[(x_1, y_1), \ldots, (x_N, y_N)|x_1, \ldots, x_N] = \prod_{i=1}^{N} p(y_i|x_i;\theta)$$

- The Maximum Likelihood Estimate for θ is the value of θ that maximizes this expression.



Log-Likelihood Function

- Usually, we do computations with the *log-likelihood*:

$$\mathcal{L}(\theta) = \log\left(p(\mathcal{S}_{\text{train}};\theta)\right) = \sum_{i=1}^{N} \log p(y_i|x_i;\theta)$$

$$\hat{\theta} = \arg\max_{\theta} \mathcal{L}(\theta)$$

- Why is this a good idea?
  - Sums are often just "easier" to work with.
  - Numerical stability is often better.

Statistical view of linear regression

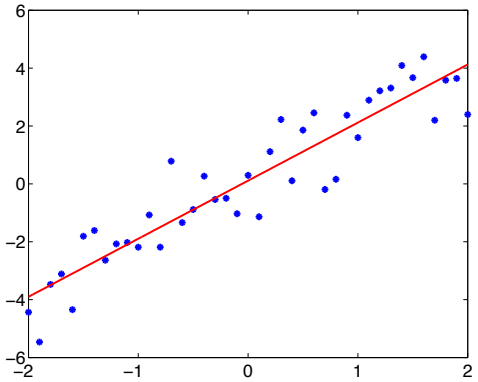
- In a statistical regression model we model both the function and noise

$$\textbf{Observed output} \;=\; \textbf{function} + \textbf{noise}$$

$$y \;=\; f(\mathbf{x}; \mathbf{w}) + \epsilon$$

where, e.g., $\epsilon \sim N(0, \sigma^2)$.

- Whatever we cannot capture with our chosen family of functions will be *interpreted* as noise

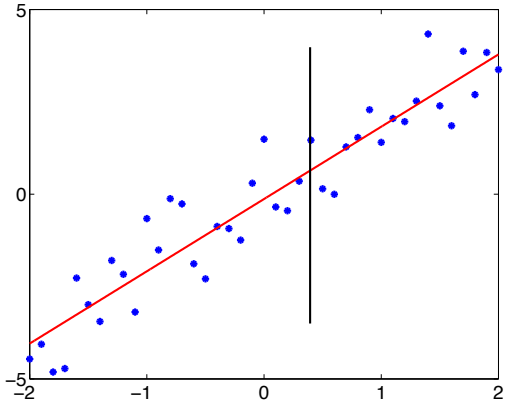


Statistical view of linear regression

- $f(\mathbf{x}; \mathbf{w})$ is trying to capture the mean of the observations y given the input $\mathbf{x}$:

$$\begin{aligned} E\{\, y \,|\, \mathbf{x} \,\} &= E\{\, f(\mathbf{x}; \mathbf{w}) + \epsilon \,|\, \mathbf{x} \,\} \\ &= f(\mathbf{x}; \mathbf{w}) \end{aligned}$$

where $E\{\, y \,|\, \mathbf{x} \,\}$ is the conditional expectation of y given $\mathbf{x}$, evaluated according to the model (not according to the underlying distribution P)



Statistical view of linear regression

- According to our statistical model

$$y = f(\mathbf{x}; \mathbf{w}) + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

the outputs y given $\mathbf{x}$ are normally distributed with mean $f(\mathbf{x}; \mathbf{w})$ and variance σ^2:

$$p(y|\mathbf{x}, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{ -\frac{1}{2\sigma^2}(y - f(\mathbf{x}; \mathbf{w}))^2 \}$$

(we model the uncertainty in the predictions, not just the mean)

- Loss function? Estimation?

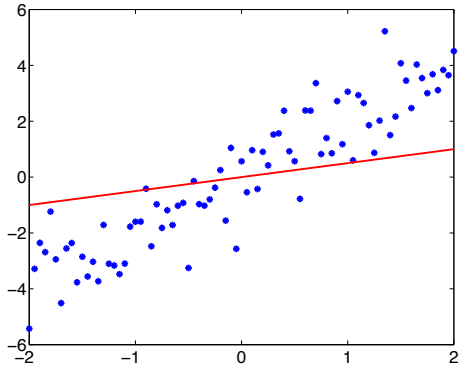
Maximum likelihood estimation

- Given observations $D_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ we find the parameters $\mathbf{w}$ that maximize the (conditional) likelihood of the outputs

$$L(D_n; \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2)$$

- Example: linear function

$$p(y|\mathbf{x}, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{ -\frac{1}{2\sigma^2}(y - w_0 - w_1 x)^2 \}$$



(why is this a bad fit according to the likelihood criterion?)

Maximum likelihood estimation (cont'd)

Likelihood of the observed outputs:

$$L(D; \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2)$$

- It is often easier (but equivalent) to try to maximize the log-likelihood:

$$\begin{aligned} l(D; \mathbf{w}, \sigma^2) &= \log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) \\ &= \sum_{i=1}^{n} \left(-\frac{1}{2\sigma^2}(y_i - f(\mathbf{x}_i; \mathbf{w}))^2 - \log \sqrt{2\pi\sigma^2} \right) \\ &= \left(-\frac{1}{2\sigma^2} \right) \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 + \dots \end{aligned}$$

Maximum likelihood estimation (cont'd)

- Maximizing log-likelihood is equivalent to minimizing empirical loss when the loss is defined according to

$$\text{Loss}(y_i, f(\mathbf{x}_i; \mathbf{w})) = -\log P(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2)$$

Loss defined as the negative log-probability is known as the log-loss.

Maximum likelihood estimation (cont'd)

- The log-likelihood of observations

$$\log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^{n} \log P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2)$$

  is a generic fitting criterion and can be used to estimate the noise variance σ^2 as well.

- Let $\hat{\mathbf{w}}$ be the maximum likelihood (here least squares) setting of the parameters. What is the maximum likelihood estimate of σ^2, obtained by solving

$$\frac{\partial}{\partial \sigma^2} \log L(D; \mathbf{w}, \sigma^2) = 0 \ ?$$

Maximum likelihood estimation (cont'd)

- The log-likelihood of observations

$$\log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2)$$

  is a generic fitting criterion and can be used to estimate the noise variance σ^2 as well.

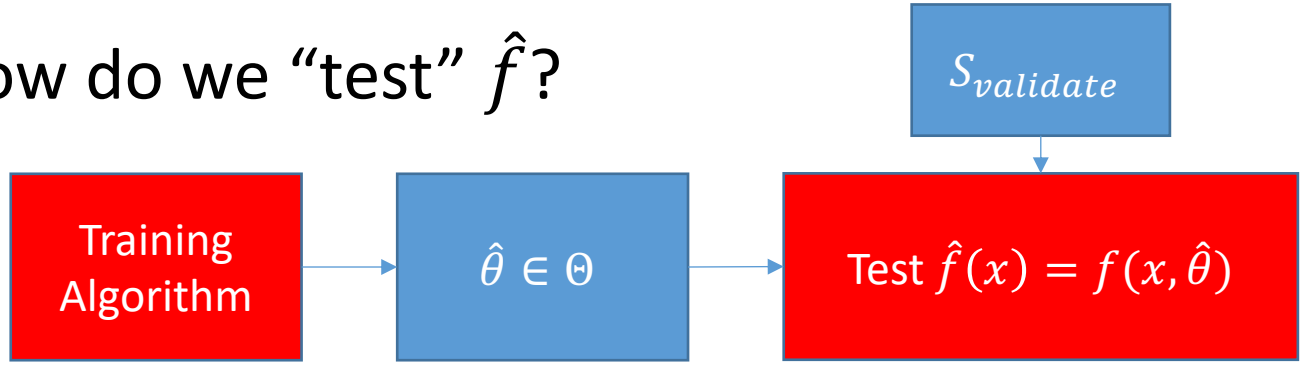
- Let $\hat{\mathbf{w}}$ be the maximum likelihood (here least squares) setting of the parameters. The maximum likelihood estimate of the noise variance σ^2 is

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \hat{\mathbf{w}}))^2$$

  i.e., the mean squared prediction error.

Empirical Risk Minimization

- ERM is the most popular and flexible way to train models. Maximum Likelihood is actually a special case of ERM.
- How do we "test" $\hat{f}$?



- Use a *cost or "risk" function* $\ell(x, y, \theta)$.

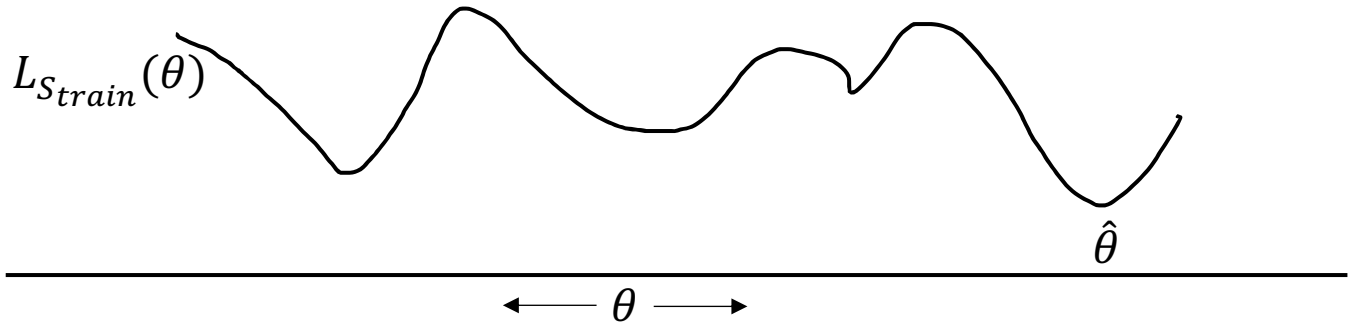
$$L_S(\theta) = \frac{1}{|S|} \sum_{(x,y) \in S} \ell(x, y, \theta)$$

Empirical Risk Minimization

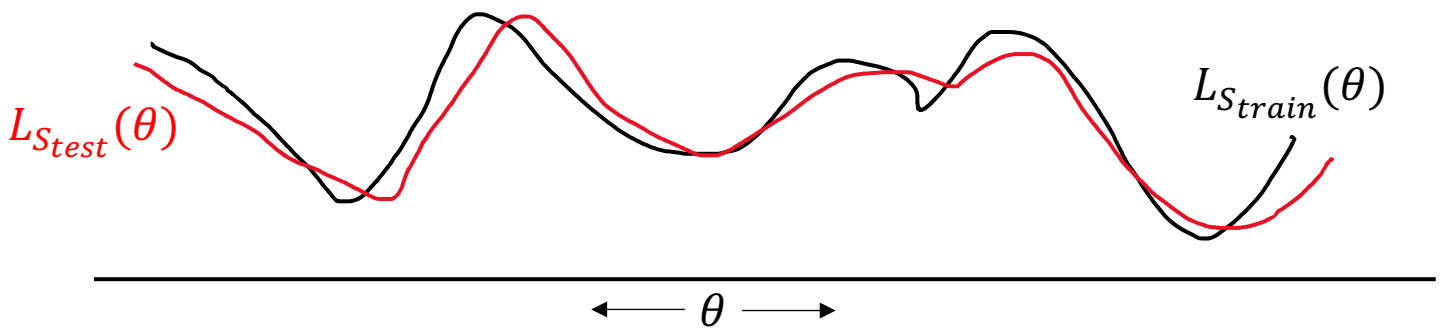
- The ERM principle says that we should pick $\hat{\theta}$ by minimizing the *empirical risk* $L_{S_{train}}(\theta)$:

$$\hat{\theta} = argmin\, L_{S_{train}}(\theta)$$

- This is also known as minimizing the training error, or often just "training", since it is so common.



$$L_{S_{train}} \approx L_{S_{test}}$$



MAP

- Just as in MLE, suppose our hypothesis class is a set of distributions:

$$H = \{p(y|x;\theta): \theta \in \Theta\}$$

- Instead of choosing $\hat{\theta}$ via the MLE, let us instead assume that the *true* value of θ was drawn from some *prior* distribution $\pi(\theta)$.
- Informally, god (or whatever process created your dataset) rolled some dice, picked a value for θ, and then used $p(y|x;\theta)$ to generate the dataset.

MAP

- Since we assume knowledge of the prior distribution $\pi(\theta)$, we actually compute the *posterior* distribution of θ given the dataset
$$\pi(\theta|S_{train})$$
- Then we choose the value of θ that maximizes this posterior distribution: the *most likely* value of θ.
- In the homework, you show this is the same as:
$$\hat{\theta} = argmax\ \mathcal{L}(\theta) + \log \pi(\theta)$$
- Where $\mathcal{L}(\theta)$ is the log likelihood function. This is analogous to adding a regularizer to an ERM loss