10-301/601: Introduction to Machine Learning Lecture 20 – Exam 2 Review

Henry Chai

7/18/22

Front Matter

- Announcements:
  - Exam 2 on 7/19 (tomorrow!)
    - Please show up to PH 100 (in-person) at 1:50 PM as the exam will begin promptly at 2 PM

3 Inference Questions for HMMs

1. Marginal Computation: $P\left(Y_t = s_j \middle| \boldsymbol{x}^{(n)}\right)$ (or $P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$)

$$P\left(Y \middle| \boldsymbol{x}^{(n)}\right) = \frac{P\left(\boldsymbol{x}^{(n)} | Y\right) P(Y)}{P\left(\boldsymbol{x}^{(n)}\right)} = \frac{\prod_{t=1}^{T} P\left(\boldsymbol{x}_t^{(n)} \middle| Y_t\right) P(Y_t | Y_{t-1})}{P\left(\boldsymbol{x}^{(n)}\right)}$$

2. Viterbi Decoding: $\hat{Y} = \underset{Y}{\operatorname{argmax}}\ P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$

3. Evaluation: $P\left(\boldsymbol{x}^{(n)}\right)$

$$P\left(\boldsymbol{x}^{(n)}\right) = \sum_{\mathcal{Y} \in \{\text{all possible sequences}\}} P\left(\boldsymbol{x}^{(n)} \middle| \mathcal{Y}\right) P(\mathcal{Y})$$

~~3~~ 4 Inference Questions for HMMs

1. Marginal Computation: $P\left(Y_t = s_j \middle| \boldsymbol{x}^{(n)}\right)$ (or $P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$)

$$P\left(Y \middle| \boldsymbol{x}^{(n)}\right) = \frac{P\left(\boldsymbol{x}^{(n)} | Y\right) P(Y)}{P\left(\boldsymbol{x}^{(n)}\right)} = \frac{\prod_{t=1}^{T} P\left(\boldsymbol{x}_t^{(n)} \middle| Y_t\right) P(Y_t | Y_{t-1})}{P\left(\boldsymbol{x}^{(n)}\right)}$$

2. Viterbi Decoding: $\hat{Y} = \underset{Y}{\operatorname{argmax}}\ P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$

3. Evaluation: $P\left(\boldsymbol{x}^{(n)}\right)$

$$P\left(\boldsymbol{x}^{(n)}\right) = \sum_{\mathcal{Y} \in \{\text{all possible sequences}\}} P\left(\boldsymbol{x}^{(n)} \middle| \mathcal{Y}\right) P(\mathcal{Y})$$

4. Minimum Bayes Risk (MBR) Decoding:

$$\hat{Y} = \underset{Y}{\operatorname{argmin}}\ \mathbb{E}_{Y' \sim P_{A,B}\left(\cdot \middle| \boldsymbol{x}^{(n)}\right)}[\ell(Y, Y')]$$

Minimum Bayes Risk Decoding

- The learned parameters A and B induce a probability distribution or belief over sequences of states $P_{A,B}\left(Y \middle| \boldsymbol{x}^{(n)}\right)$
- Given a loss function, $\ell(Y, Y')$, find the sequence of states that minimizes our expected loss *under our current belief*

$$\hat{Y} = \underset{Y}{\operatorname{argmin}} \ \mathbb{E}_{Y' \sim P_{A,B}\left(\cdot \middle| \boldsymbol{x}^{(n)}\right)}[\ell(Y, Y')]$$

$$= \underset{Y}{\operatorname{argmin}} \sum_{Y'} P_{A,B}\left(Y' \middle| \boldsymbol{x}^{(n)}\right) \ell(Y, Y')$$

Minimum Bayes Risk Decoding: Example

- If $\ell(Y, Y')$ is the 0-1 loss

$$\ell(Y, Y') = 1 - \mathbb{1}(Y = Y')$$

$$\hat{Y} = \underset{Y}{\operatorname{argmin}} \sum_{Y'} P_{A,B}\left(Y' \middle| \boldsymbol{x}^{(n)}\right)\left(1 - \mathbb{1}(Y = Y')\right)$$

$$= \underset{Y}{\operatorname{argmin}} - \sum_{Y'} P_{A,B}\left(Y' \middle| \boldsymbol{x}^{(n)}\right) \mathbb{1}(Y = Y')$$

$$= \underset{Y}{\operatorname{argmax}}\ P_{A,B}\left(Y \middle| \boldsymbol{x}^{(n)}\right)$$

Minimum Bayes Risk Decoding: Example

- If $\ell(Y, Y')$ is the Hamming loss

$$\ell(Y, Y') = \sum_{t=1}^{T} 1 - \mathbb{1}(Y_t = Y_t')$$

$$\hat{Y}_t = \underset{Y_t}{\operatorname{argmax}} \; P_{A,B}\left(Y_t \middle| \boldsymbol{x}^{(n)}\right)$$

- Computes the most likely state at each time step using marginals

Key Takeaways

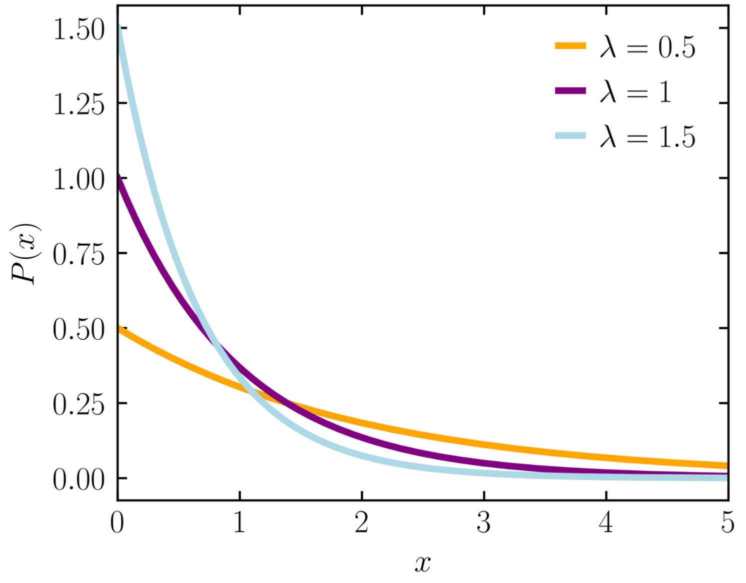
- HMMs are an instantiation of (dynamic) Bayesian networks where certain parameters are shared
  - Parameters can be set by MLE
- Because of their well-behaved graphical structure, inference in HMMs is tractable via dynamic programming
  - Forward-backward algorithm for computing marginal distributions
  - Viterbi algorithm for computing most probable sequence of states

Probabilistic Learning

- Previously:
  - (Unknown) Target function, $c^*: \mathcal{X} \rightarrow \mathcal{Y}$
  - Classifier, $h : \mathcal{X} \rightarrow \mathcal{Y}$
  - Goal: find a classifier, h, that best approximates c^*
- Now:
  - (Unknown) Target *distribution*, $y \sim p^*(Y|\boldsymbol{x})$
  - Distribution, $p(Y|\boldsymbol{x})$
  - Goal: find a distribution, p, that best approximates p^*

Maximum Likelihood Estimation (MLE)

- Insight: every valid probability distribution has a finite amount of probability mass as it must sum/integrate to 1
- Idea: set the parameter(s) so that the likelihood of the samples is maximized
- Intuition: assign as much of the (finite) probability mass to the observed data *at the expense of unobserved data*
- Example: the exponential distribution



Source: https://en.wikipedia.org/wiki/Exponential_distribution#/media/File:Exponential_probability_density.svg

Exponential Distribution MLE

- The pdf of the exponential distribution is

$$f(x|\lambda) = \lambda e^{-\lambda x}$$

- Given N iid samples $\{x^{(1)}, \dots, x^{(N)}\}$, the log-likelihood is

$$\ell(\lambda) = \sum_{n=1}^{N} \log f\left(x^{(n)}|\lambda\right) = \sum_{n=1}^{N} \log \lambda e^{-\lambda x^{(n)}}$$

$$= \sum_{n=1}^{N} \log \lambda + \log e^{-\lambda x^{(n)}} = N \log \lambda - \lambda \sum_{n=1}^{N} x^{(n)}$$

- Taking the partial derivative and setting it equal to 0 gives

$$\frac{\partial \ell}{\partial \lambda} = \frac{N}{\lambda} - \sum_{n=1}^{N} x^{(n)}$$

Exponential Distribution MLE

- The pdf of the exponential distribution is
$$f(x|\lambda) = \lambda e^{-\lambda x}$$
- Given N iid samples $\{x^{(1)}, \ldots, x^{(N)}\}$, the log-likelihood is
$$\ell(\lambda) = \sum_{n=1}^{N} \log f\left(x^{(n)}|\lambda\right) = \sum_{n=1}^{N} \log \lambda e^{-\lambda x^{(n)}}$$
$$= \sum_{n=1}^{N} \log \lambda + \log e^{-\lambda x^{(n)}} = N \log \lambda - \lambda \sum_{n=1}^{N} x^{(n)}$$
- Taking the partial derivative and setting it equal to 0 gives
$$\frac{N}{\hat{\lambda}} - \sum_{n=1}^{N} x^{(n)} = 0 \rightarrow \frac{N}{\hat{\lambda}} = \sum_{n=1}^{N} x^{(n)} \rightarrow \hat{\lambda} = \frac{N}{\sum_{n=1}^{N} x^{(n)}}$$

Practice Problem: MLE

- The pdf of the Gamma distribution is

$$f(x|\alpha,\beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

- Given N iid samples $\{x^{(1)}, \dots, x^{(N)}\}$, what is the MLE of β?

$$\ell(\alpha,\beta) = \sum_{n=1}^{N} \log f\left(x^{(n)}|\alpha,\beta\right) = \sum_{n=1}^{N} \log \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

$$= \sum_{n=1}^{N} \alpha \log \beta - \log \Gamma(\alpha) + (\alpha - 1) \log x^{(n)} - \beta x^{(n)}$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{n=1}^{N} \frac{\alpha}{\beta} - x^{(n)} = \frac{n\alpha}{\beta} - \sum_{n=1}^{N} x^{(n)}$$

$$\rightarrow \frac{n\alpha}{\hat{\beta}} - \sum_{n=1}^{N} x^{(n)} = 0 \rightarrow \hat{\beta} = \frac{n\alpha}{\sum_{n=1}^{N} x^{(n)}}$$

Maximum a Posteriori (MAP) Estimation

- Insight: sometimes we have *prior* information we want to incorporate into parameter estimation
- Idea: use Bayes rule to reason about the *posterior* distribution over the parameters
  - MLE finds $\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{D}|\theta)$
  - MAP finds $\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\theta|\mathcal{D})$

$$= \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{D}|\theta)p(\theta)/p(\mathcal{D})$$

$$= \underset{\theta}{\operatorname{argmax}}\ \underbrace{p(\mathcal{D}|\theta)}_{\text{likelihood}}\underbrace{p(\theta)}_{\text{prior}}$$

$$= \underset{\theta}{\operatorname{argmax}}\ \underbrace{\log p(\mathcal{D}|\theta) + \log p(\theta)}_{\text{log-posterior}}$$

Coin Flipping MAP

- A Bernoulli random variable takes value 1 (or heads) with probability ϕ and value 0 (or tails) with probability $1 - \phi$
- The pmf of the Bernoulli distribution is

$$p(x|\phi) = \phi^x(1 - \phi)^{1-x}$$

- Assume a Beta prior over the parameter ϕ, which has pdf

$$f(\phi|\alpha, \beta) = \frac{\phi^{\alpha-1}(1 - \phi)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}$$

- where $\mathrm{B}(\alpha, \beta) = \int_0^1 \phi^{\alpha-1}(1 - \phi)^{\beta-1} d\phi$ is a normalizing constant to ensure the distribution integrates to 1

Coin Flipping MAP

- Given N iid samples $\{x^{(1)}, \dots, x^{(N)}\}$, the partial derivative of the log-posterior is

$$\frac{\partial \ell}{\partial \phi} = \frac{(\alpha - 1 + N_1)}{\phi} - \frac{(\beta - 1 + N_0)}{1 - \phi}$$

$$\vdots$$

$$\rightarrow \hat{\phi}_{MAP} = \frac{(\alpha - 1 + N_1)}{(\beta - 1 + N_0) + (\alpha - 1 + N_1)}$$

- $\alpha - 1$ is a "pseudocount" of the number of 1's (or heads) you've "observed"
- $\beta - 1$ is a "pseudocount" of the number of 0's (or tails) you've "observed"

Building a Probabilistic Classifier

- Define a decision rule
  - Given a test data point $\boldsymbol{x}'$, predict its label $\hat{y}$ using the posterior distribution $P(Y = y | X = \boldsymbol{x}')$
  - Common choice: $\hat{y} = \underset{y}{\operatorname{argmax}} P(Y = y | X = \boldsymbol{x}')$
- Model the posterior distribution
  - Option 1 - Model $P(Y|X)$ directly as some function of X (tomorrow)
  - Option 2 - Use Bayes' rule (today!):

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)} \propto P(X|Y)\,P(Y)$$

How hard is modelling $P(X|Y)$?

x_1 ("hat")	x_2 ("cat")	x_3 ("dog")	x_4 ("fish")	x_5 ("mom")	x_6 ("dad")	$P(X\|Y=1)$	$P(X\|Y=0)$
0	0	0	0	0	0	θ_1	θ_{64}
1	0	0	0	0	0	θ_2	θ_{65}
1	1	0	0	0	0	θ_3	θ_{66}
1	0	1	0	0	0	θ_4	θ_{67}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	1	1	1	1	$1-\sum_{i=1}^{63}\theta_i$	$1-\sum_{i=64}^{126}\theta_i$

Naïve Bayes Assumption

- ***Assume*** features are conditionally independent given the label:

$$P(X|Y) = \prod_{d=1}^{D} P(X_d|Y)$$

- Pros:
  - <u>Significantly</u> reduces computational complexity
  - Also reduces model complexity, combats overfitting
- Cons:
  - Is a strong, often illogical assumption
    - We'll see a relaxed version of this later in the semester when we discuss Bayesian networks

Recipe for Naïve Bayes

- Define a model and model parameters
  - Make the Naïve Bayes assumption
  - Assume independent, identically distributed (iid) data
  - Parameters: $\pi = P(Y = 1)$, $\theta_{d,y} = P(X_d = 1 | Y = y)$
- Write down an objective function
  - Maximize the log-likelihood
- Optimize the objective w.r.t. the model parameters
  - Solve in *closed form*: take partial derivatives, set to 0 and solve

Bernoulli Naïve Bayes

- Binary label
  - $Y \sim \text{Bernoulli}(\pi)$
  - $\hat{\pi} = {}^{N_{Y=1}}/_{N}$
    - N = # of data points
    - $N_{Y=1}$ = # of data points with label 1
- Binary features
  - $X_d | Y = y \sim \text{Bernoulli}(\theta_{d,y})$
  - $\hat{\theta}_{d,y} = {}^{N_{Y=y,\, X_d=1}}/_{N_{Y=y}}$
    - $N_{Y=y}$ = # of data points with label y
    - $N_{Y=y,\, X_d=1}$ = # of data points with label y and feature $X_d = 1$

Multinomial Naïve Bayes

- Binary label
  - $Y \sim \text{Bernoulli}(\pi)$
  - $\hat{\pi} = {}^{N_{Y=1}}/_{N}$
    - N = # of data points
    - $N_{Y=1}$ = # of data points with label 1
- Discrete features (X_d can take on one of K possible values)
  - $X_d | Y = y \sim \text{Categorical}\left(\theta_{d,1,y}, \dots, \theta_{d,K-1,y}\right)$
  - $\hat{\theta}_{d,k,y} = {}^{N_{Y=y,\, X_d=k}}/_{N_{Y=y}}$
    - $N_{Y=y}$ = # of data points with label y
    - $N_{Y=y,\, X_d=k}$ = # of data points with label y and feature $X_d = k$

Gaussian Naïve Bayes

- Binary label
  - $Y \sim \text{Bernoulli}(\pi)$
  - $\hat{\pi} = {}^{N_{Y=1}}/_{N}$
    - N = # of data points
    - $N_{Y=1}$ = # of data points with label 1
- Real-valued features
  - $X_d | Y = y \sim \text{Gaussian}\left(\mu_{d,y}, \sigma_{d,y}^2\right)$
  - $\hat{\mu}_{d,y} = \frac{1}{N_{Y=y}} \sum_{n: y^{(n)}=y} x_d^{(n)}$
  - $\hat{\sigma}_{d,y}^2 = \frac{1}{N_{Y=y}} \sum_{n: y^{(n)}=y} \left(x_d^{(n)} - \hat{\mu}_{d,y}\right)^2$
    - $N_{Y=y}$ = # of data points with label y

Practice Problem: Naïve Bayes

- Given a binary label and D discrete features, each of which can take on K possible values, how many parameters would a multinomial naïve Bayes model need to learn?

$$2\big(D(K-1)\big)+1$$

- Given a binary label and D real-valued features, how many parameters would a Gaussian naïve Bayes model need to learn?

$$2(2D)+1$$

Visualizing Gaussian Naïve Bayes (2 classes, equal variances)

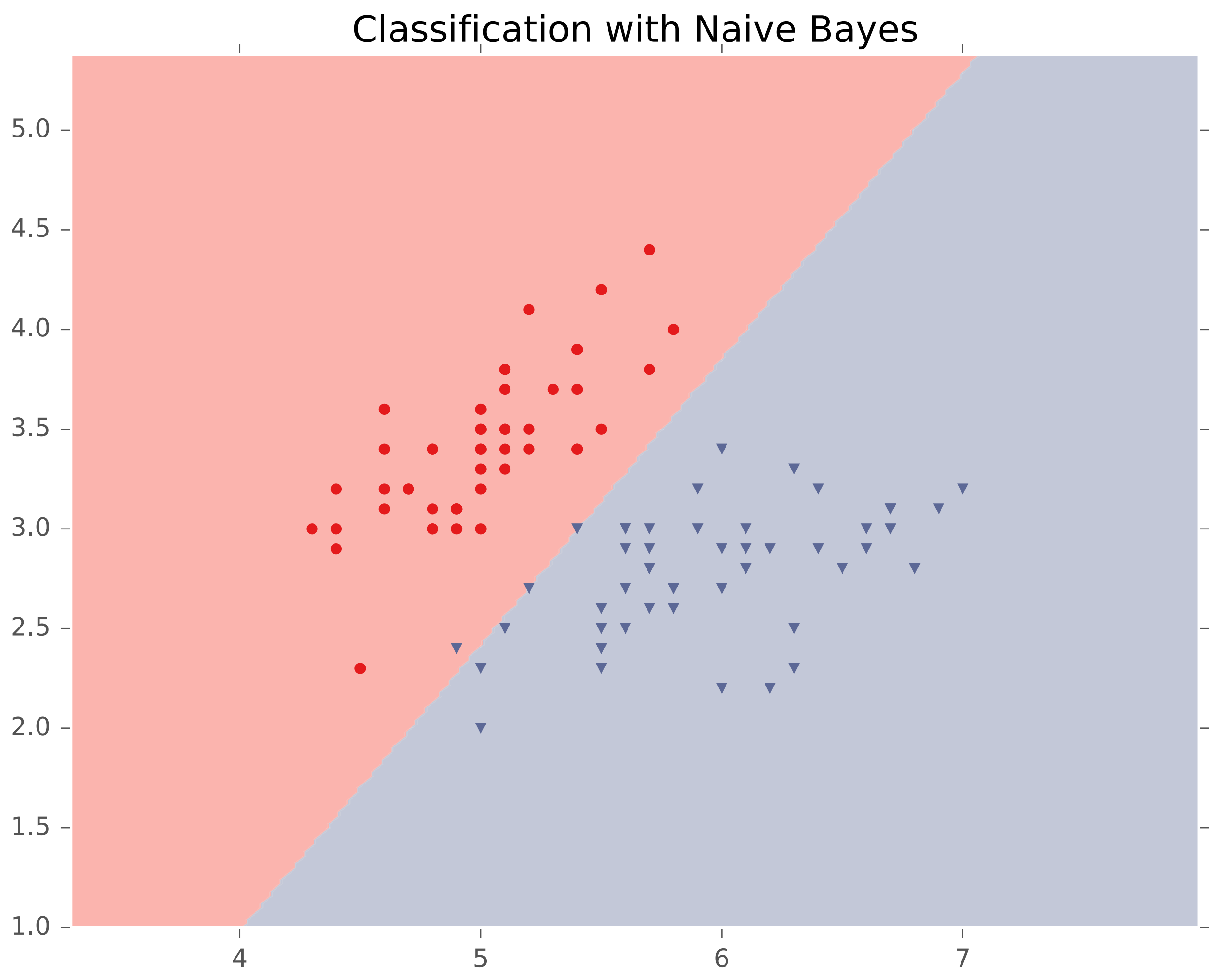


Figure courtesy of Matt Gormley

Visualizing Gaussian Naïve Bayes (2 classes, learned variances)

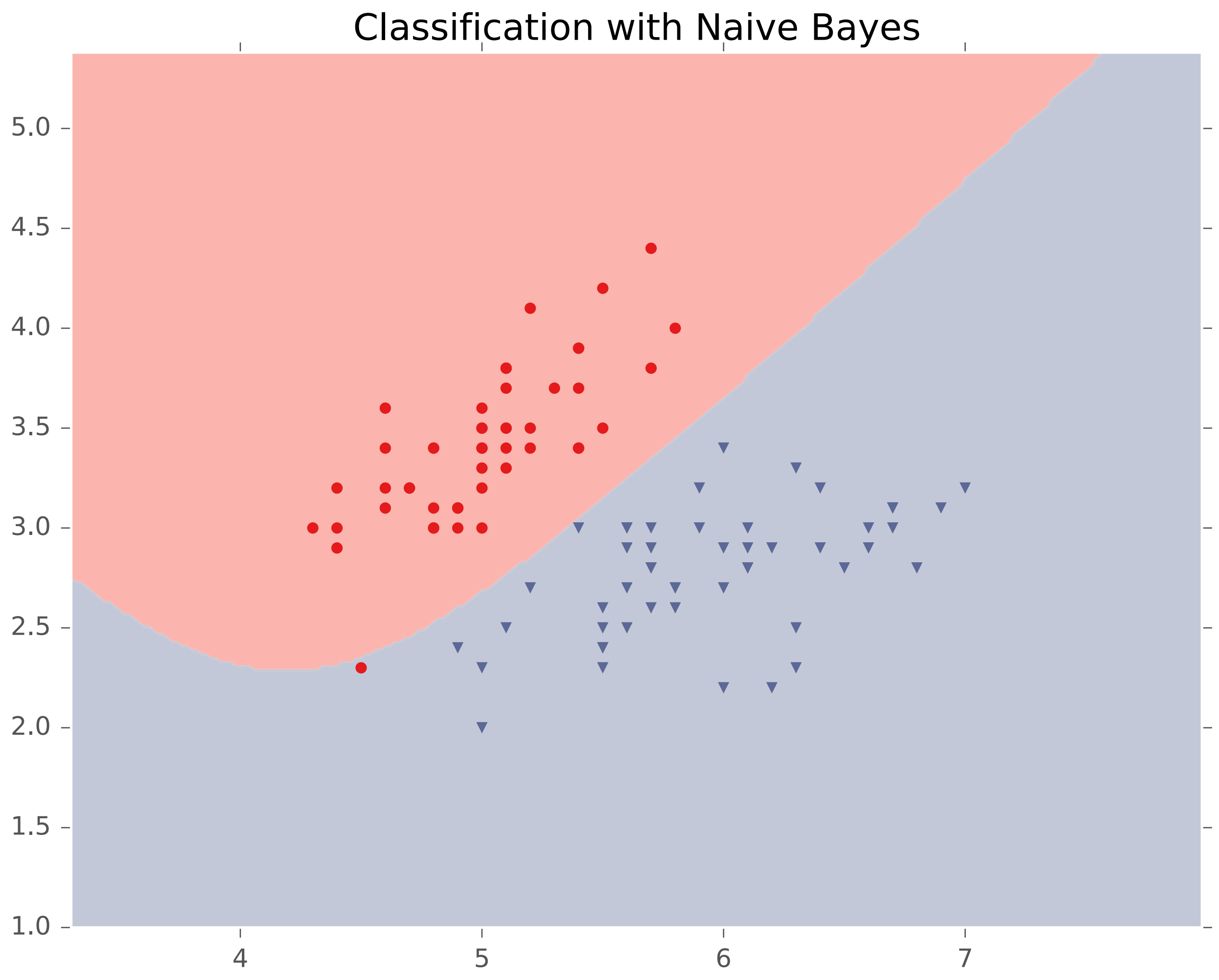


Figure courtesy of Matt Gormley

Bernoulli Naïve Bayes: Making Predictions

- Given a test data point $\boldsymbol{x}' = [x'_1, \dots, x'_D]^T$

$$P(Y = 1|\boldsymbol{x}') \propto P(Y = 1)P(\boldsymbol{x}'|Y = 1)$$
$$= \hat{\pi} \prod_{d=1}^{D} \hat{\theta}_{d,1}^{x'_d} \left(1 - \hat{\theta}_{d,1}\right)^{1-x'_d}$$

$$P(Y = 0|\boldsymbol{x}') \propto (1 - \hat{\pi}) \prod_{d=1}^{D} \hat{\theta}_{d,0}^{x'_d} \left(1 - \hat{\theta}_{d,0}\right)^{1-x'_d}$$

$$\hat{y} = \begin{cases} 1 \text{ if } \hat{\pi} \prod_{d=1}^{D} \hat{\theta}_{d,1}^{x'_d} \left(1 - \hat{\theta}_{d,1}\right)^{1-x'_d} > \\ \qquad (1 - \hat{\pi}) \prod_{d=1}^{D} \hat{\theta}_{d,0}^{x'_d} \left(1 - \hat{\theta}_{d,0}\right)^{1-x'_d} \\ 0 \text{ otherwise} \end{cases}$$

Setting the Parameters via MAP

- Binary label
    - $Y \sim \text{Bernoulli}(\pi)$
    - $\hat{\pi} = {}^{N_{Y=1}}/_{N}$
        - N = # of data points
        - $N_{Y=1}$ = # of data points with label 1
- Binary features
    - $X_d | Y = y \sim \text{Bernoulli}\left(\theta_{d,y}\right)$ and $\theta_{d,y} \sim \text{Beta}(\alpha, \beta)$
    - $\hat{\theta}_{d,y} = {}^{N_{Y=y,\, X_d=1} + (\alpha-1)}/_{N_{Y=y} + (\alpha-1) + (\beta-1)}$
        - $N_{Y=y}$ = # of data points with label y
        - $N_{Y=y,\, X_d=1}$ = # of data points with label y and feature $X_d = 1$
    - Common choice: $\alpha = 2, \beta = 2$

Building a Probabilistic Classifier

- Define a decision rule
  - Given a test data point $\boldsymbol{x}'$, predict its label $\hat{y}$ using the posterior distribution $P(Y = y | X = \boldsymbol{x}')$
  - Common choice: $\hat{y} = \underset{y}{\operatorname{argmax}}\, P(Y = y | X = \boldsymbol{x}')$
- Model the posterior distribution
  - Option 1 - Model $P(Y|X)$ directly as some function of X (~~tomorrow~~ today!)
  - Option 2 - Use Bayes' rule (~~today!~~ yesterday):

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)} \propto P(X|Y)\,P(Y)$$

Modelling the Posterior

- Suppose we have binary labels $y \in \{0,1\}$ and D-dimensional inputs $\boldsymbol{x} = [1, x_1, \dots, x_D]^T \in \mathbb{R}^{D+1}$
- **Assume**

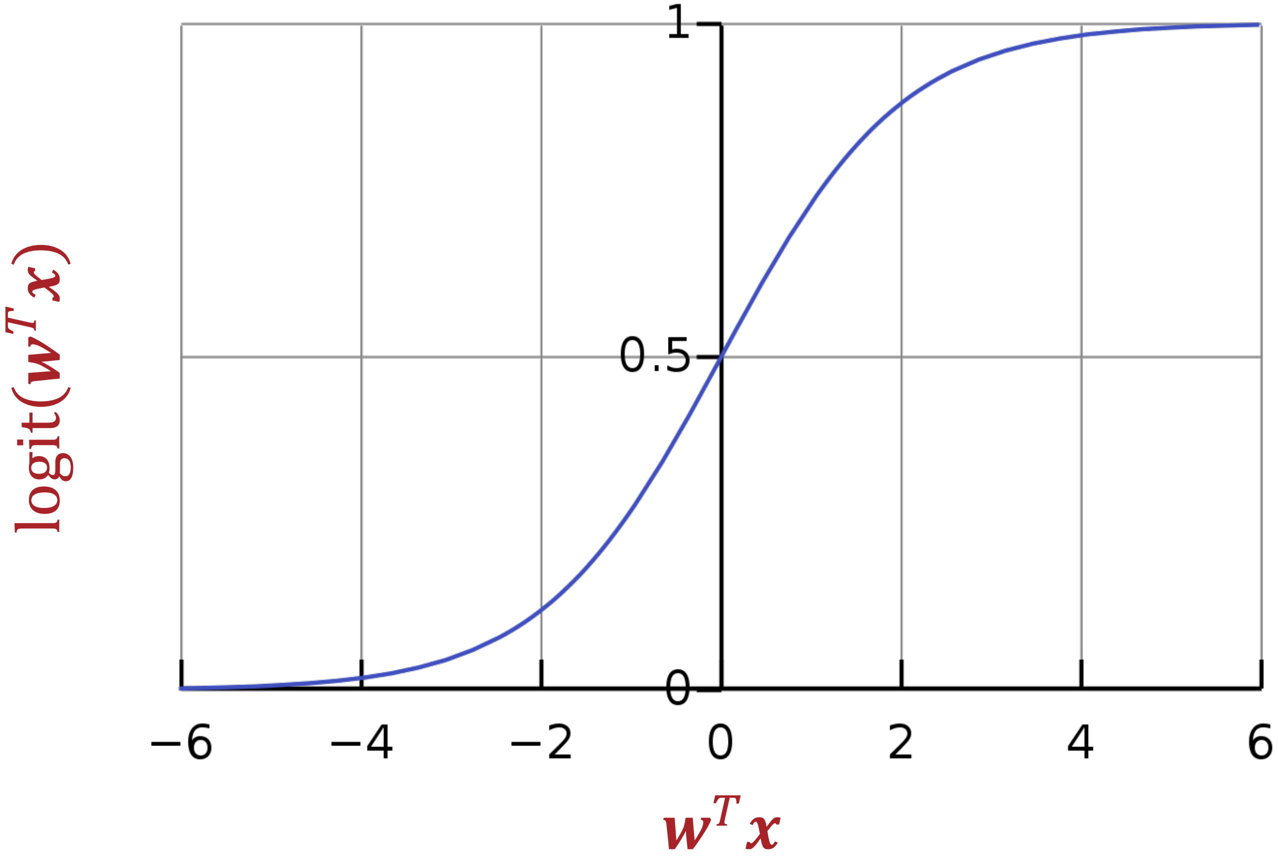
$$P(Y = 1|\boldsymbol{x}) = \text{logit}(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x})}$$

$$= \frac{\exp(\boldsymbol{w}^T\boldsymbol{x})}{\exp(\boldsymbol{w}^T\boldsymbol{x}) + 1}$$

- This implies two useful facts:

1. $P(Y = 0|\boldsymbol{x}) = 1 - P(Y = 1|\boldsymbol{x}) = \dfrac{1}{\exp(\boldsymbol{w}^T\boldsymbol{x}) + 1}$
2. $\dfrac{P(Y = 1|\boldsymbol{x})}{P(Y = 0|\boldsymbol{x})} = \exp(\boldsymbol{w}^T\boldsymbol{x}) \rightarrow \log \dfrac{P(Y = 1|\boldsymbol{x})}{P(Y = 0|\boldsymbol{x})} = \boldsymbol{w}^T\boldsymbol{x}$

Why use the Logistic Function?



- Differentiable everywhere
- $\text{logit}: \mathbb{R} \rightarrow [0, 1]$
- The decision boundary is linear in $\boldsymbol{x}$!

Source: https://en.wikipedia.org/wiki/Logistic_function#/media/File:Logistic-curve.svg

Logistic Regression Decision Boundary

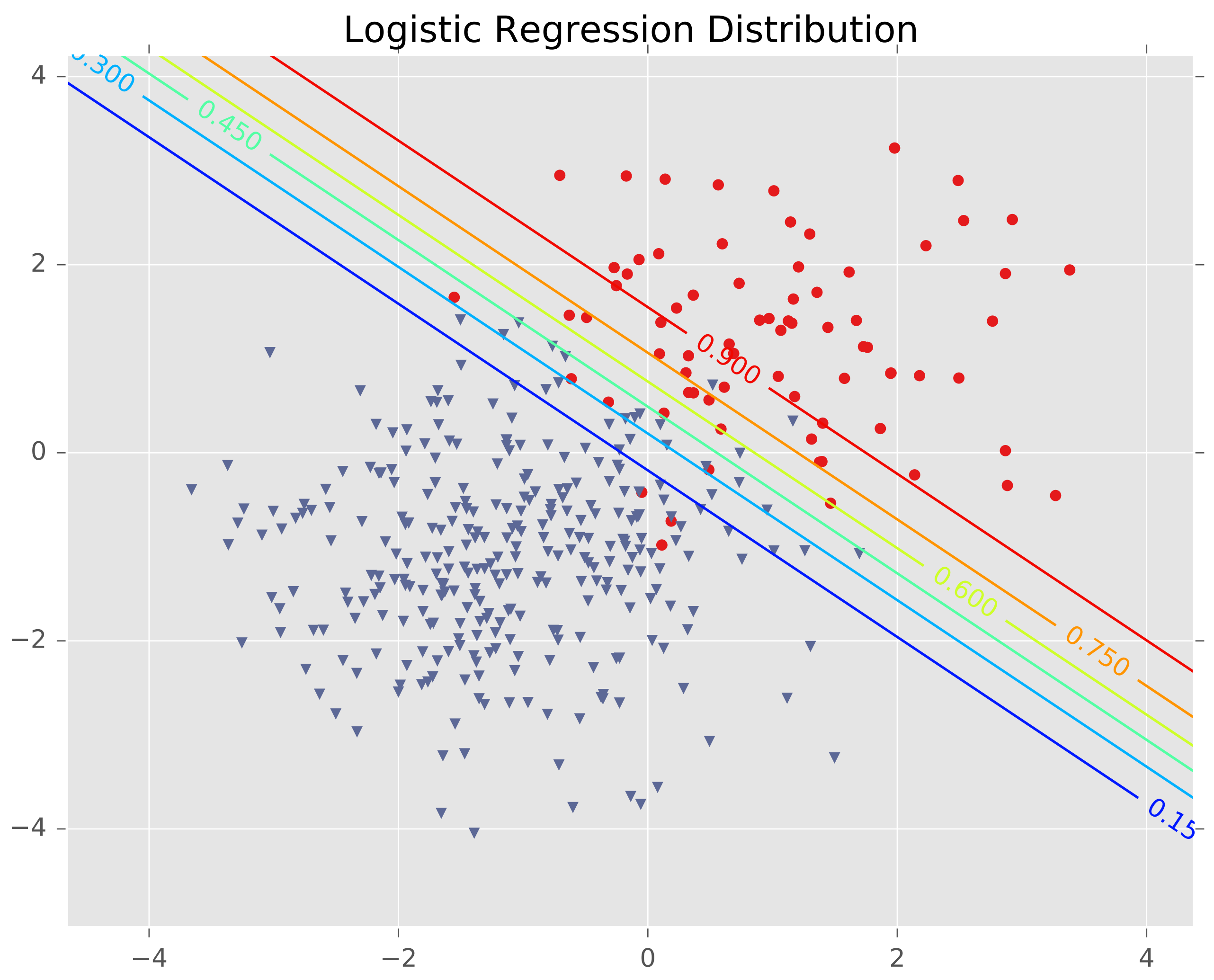


Figure courtesy of Matt Gormley

Logistic Regression Decision Boundary

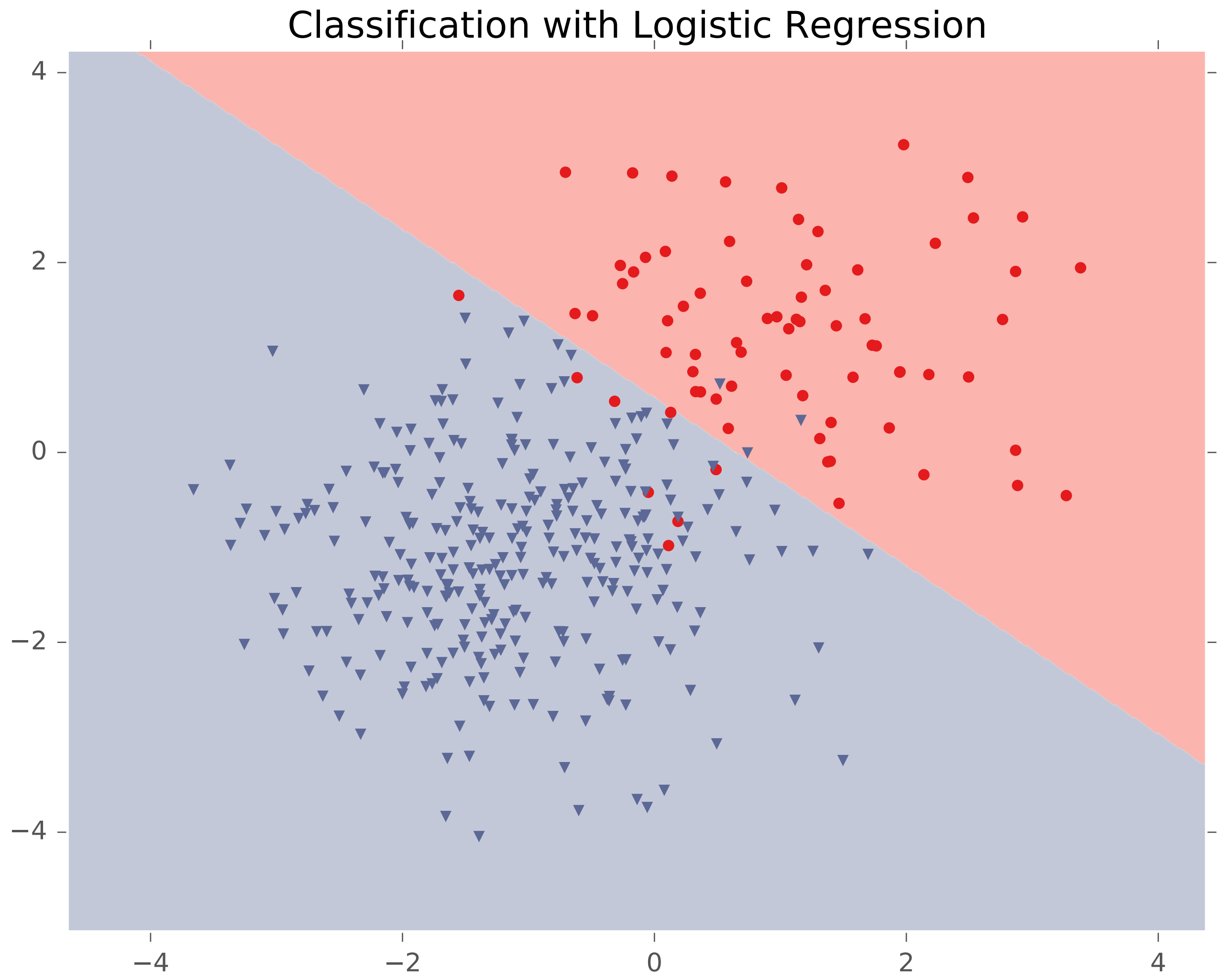


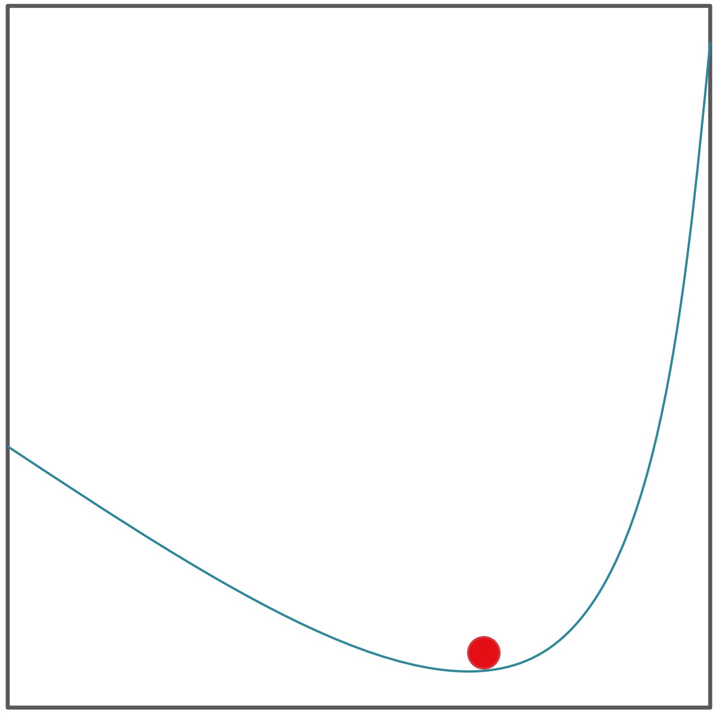
Figure courtesy of Matt Gormley

Recipe for Logistic Regression

- Define a model and model parameters
  - Assume independent, identically distributed (iid) data
  - Assume $P(Y = 1|X) = \text{logit}(\boldsymbol{w}^T \boldsymbol{x})$
  - Parameters: $\boldsymbol{w} = [w_0, w_1, \ldots, w_D]$
- Write down an objective function
  - ~~Maximize the *conditional* log-likelihood~~
  - Minimize the negative conditional log-likelihood
- Optimize the objective w.r.t. the model parameters
  - ???

Recall: Gradient Descent

- An iterative method for minimizing functions
- Requires the gradient to exist everywhere



- Good news: the negative conditional log-likelihood is convex! (See HW/recitation)

Gradient Descent

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta^{(0)}$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
    a. Compute the gradient:

$$O(ND)\left\{ \nabla_{\boldsymbol{w}} \ell_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right) = \sum_{n=1}^{N} \boldsymbol{x}^{(n)}\left(P\left(Y = 1 \middle| \boldsymbol{x}^{(n)}, \boldsymbol{w}^{(t)}\right) - y^{(n)}\right) \right.$$

    b. Update $\boldsymbol{w}$: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta^{(0)} \nabla_{\boldsymbol{w}} \ell_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right)$
    c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

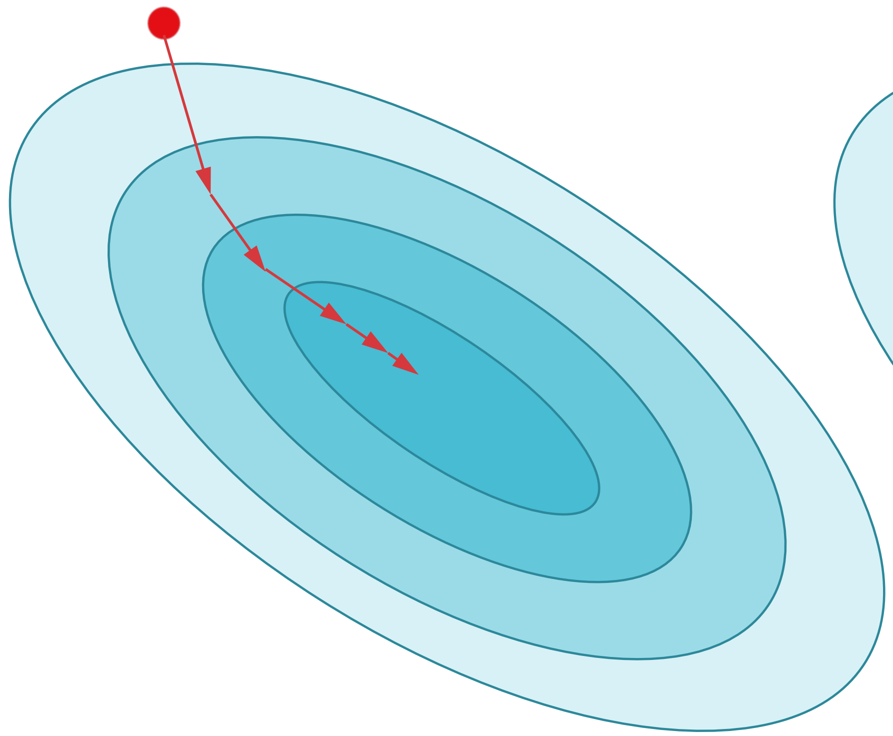
Stochastic Gradient Descent

- Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta_{SGD}^{(0)}$

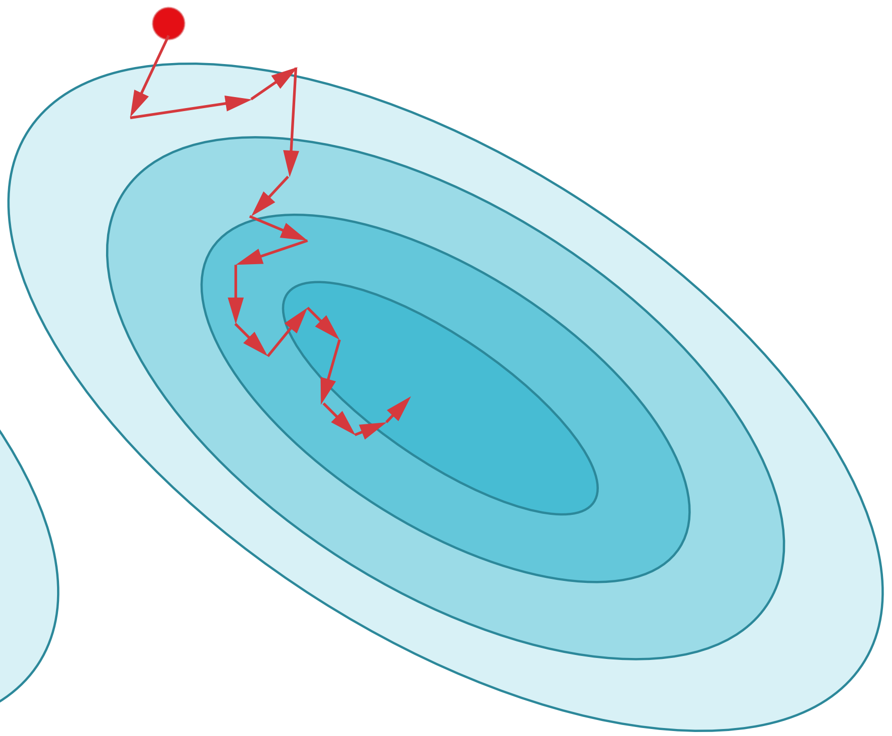
1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample a data point from $\mathcal{D}$, $(\boldsymbol{x}^{(n)}, y^{(n)})$
   b. Compute the pointwise gradient:
      $\nabla_{\boldsymbol{w}} \ell_{\boldsymbol{x}^{(n)}, y^{(n)}}(\boldsymbol{w}^{(t)}) = \boldsymbol{x}^{(n)}\left(P\left(Y = 1 \middle| \boldsymbol{x}^{(n)}, \boldsymbol{w}^{(t)}\right) - y^{(n)}\right)$
   c. Update $\boldsymbol{w}$: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta_{SGD}^{(0)} \nabla_{\boldsymbol{w}} \ell_{\boldsymbol{x}^{(n)}, y^{(n)}}(\boldsymbol{w}^{(t)})$
   d. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Stochastic Gradient Descent vs. Gradient Descent



Gradient Descent



Stochastic Gradient Descent

Mini-batch Stochastic Gradient Descent

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
    a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
    b. Compute the gradient w.r.t. the sampled *batch*:

$$\nabla_{\boldsymbol{w}} \ell_{\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}}\left(\boldsymbol{w}^{(t)}\right) = \sum_{b=1}^{B} \boldsymbol{x}^{(b)}\left(P\left(Y = 1 \middle| \boldsymbol{x}^{(b)}, \boldsymbol{w}\right) - y^{(b)}\right)$$

    c. Update $\boldsymbol{w}$: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta_{MB}^{(0)} \nabla_{\boldsymbol{w}} \ell_{\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}}\left(\boldsymbol{w}^{(t)}\right)$
    d. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Logistic Regression vs. Naïve Bayes

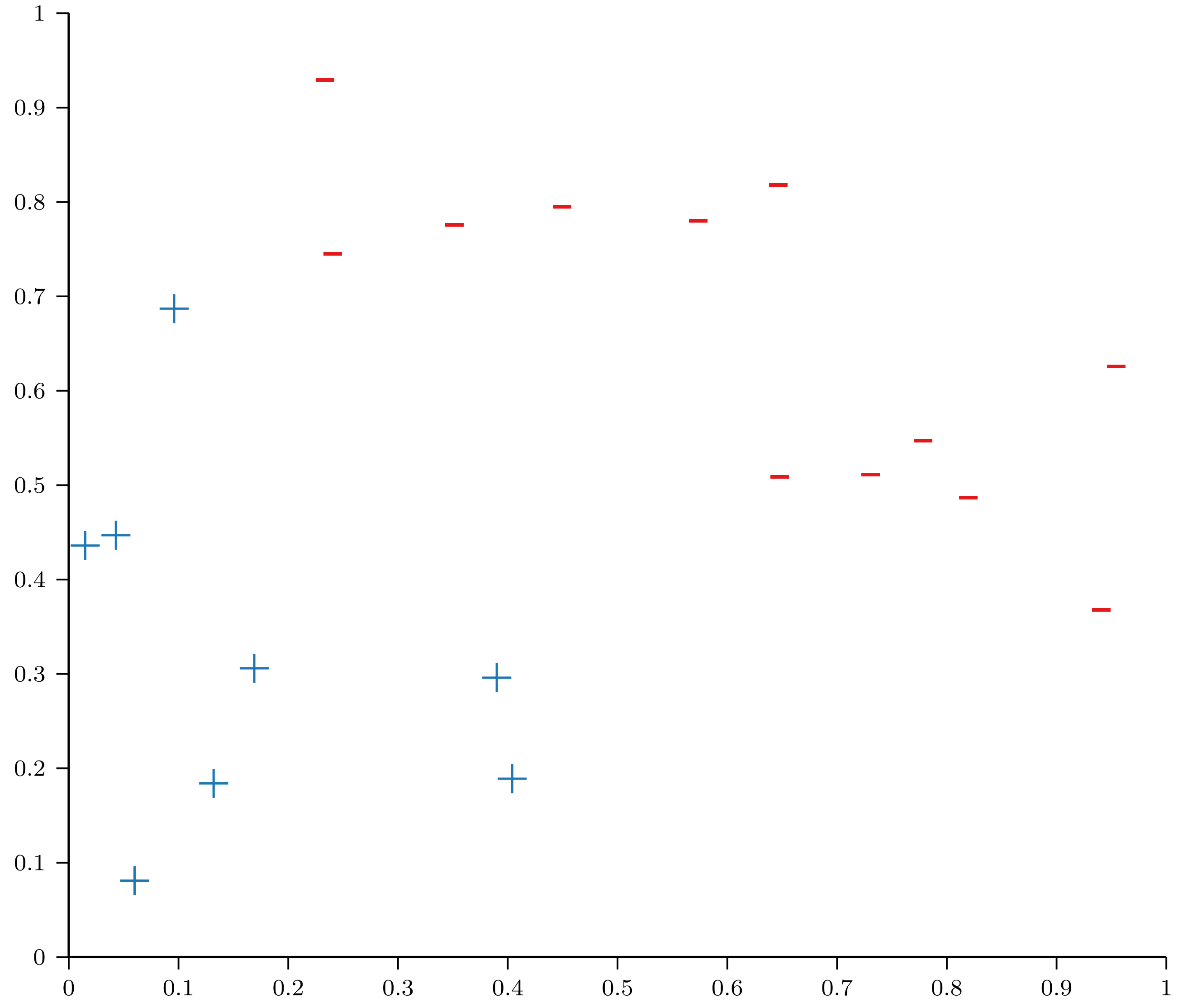
- Naïve Bayes is a *generative* model
  - By modelling $P(X|Y)$ and $P(Y)$, we can *generate* new data points:
    1. Sample a label $y \sim P(Y)$
    2. Sample features $x_d \sim P(X_d|Y = y)$
- Logistic regression is a *discriminative* model
  - By modelling $P(Y|X)$, we can only *discriminate* (or distinguish) between classes.

Logistic Regression vs. Naïve Bayes (Ng and Jordan, 2001)

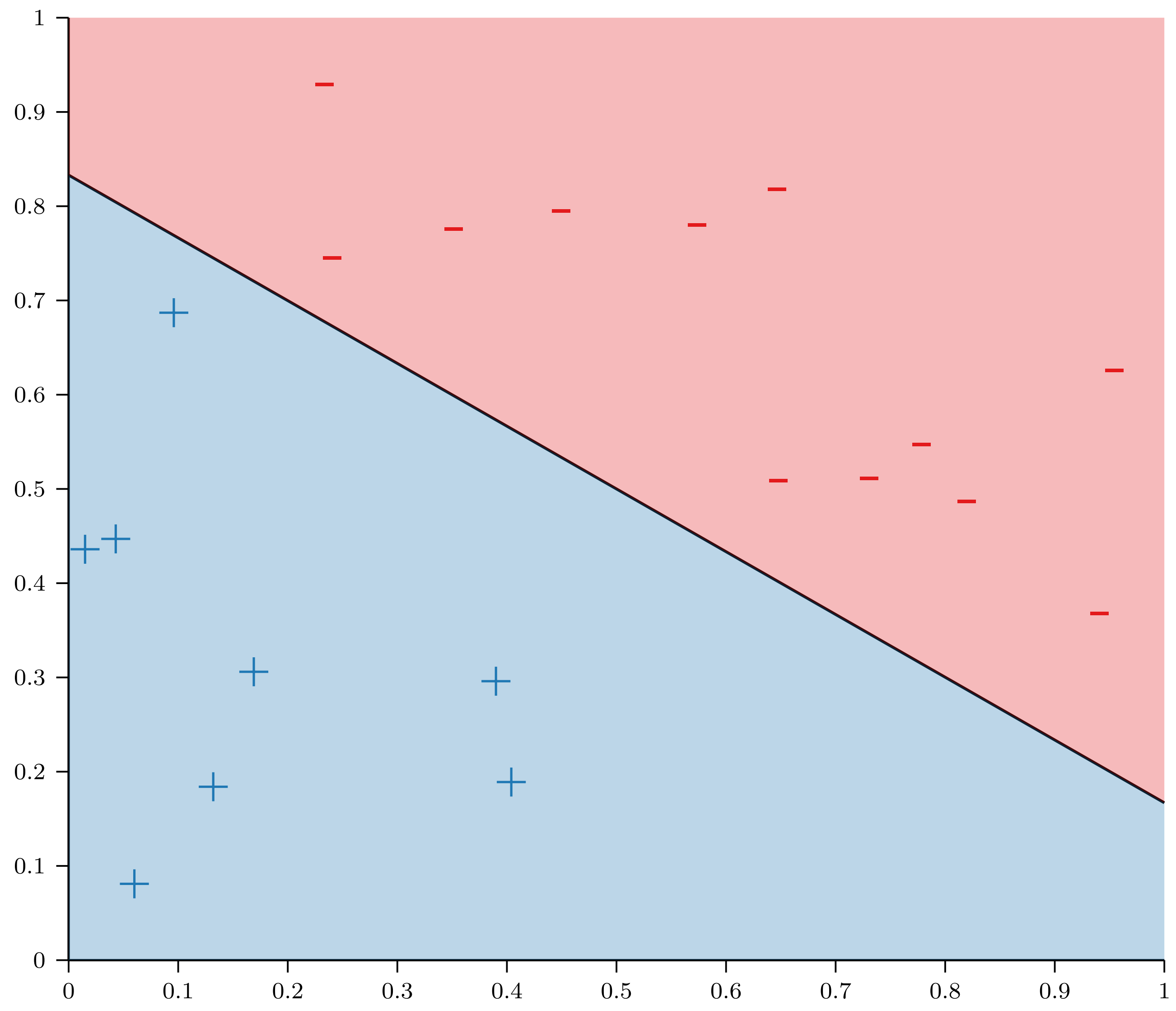
- Naïve Bayes and logistic regression form a *generative-discriminative* model pair
  - Recall that under certain conditions, the Gaussian Naïve Bayes (GNB) decision boundary is linear
  - If the Naïve Bayes assumption holds, then in the limit of infinite training data, GNB and logistic regression learn the same (linear) decision boundary!
  - In general, Naïve Bayes performs well when data is scarce but logistic regression has lower asymptotic error.

Source: http://robotics.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf

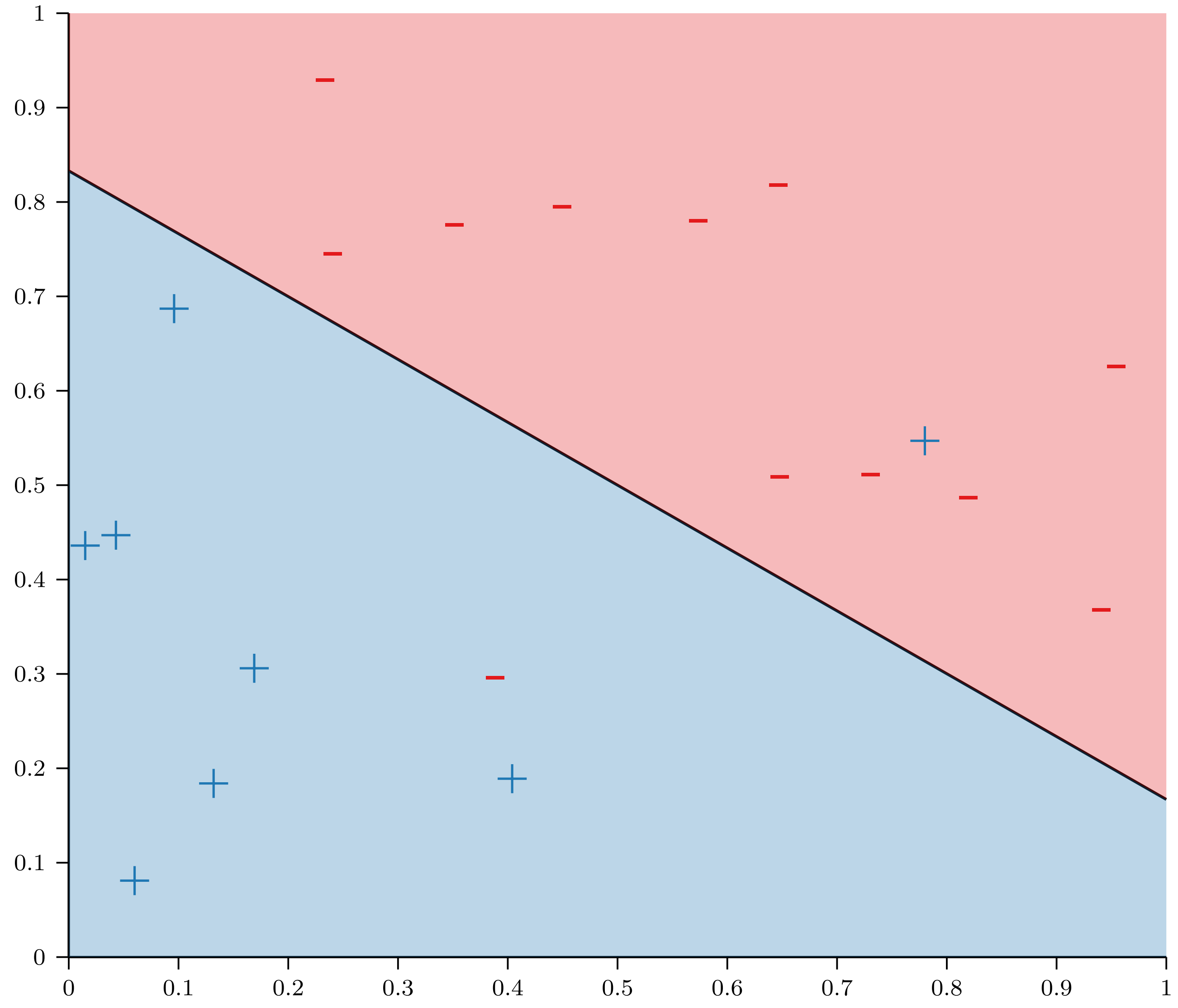
Linear Models



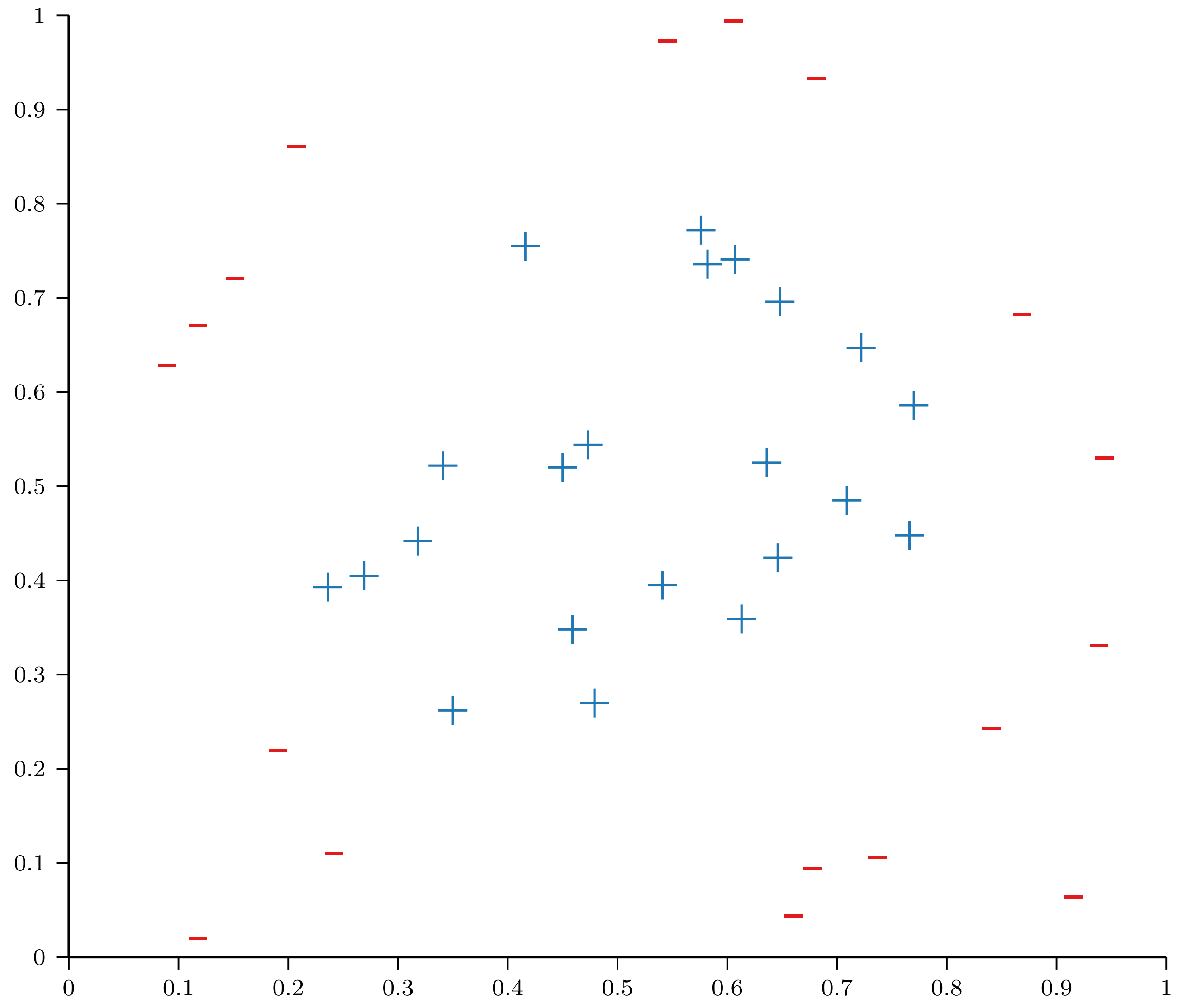
Linear Models



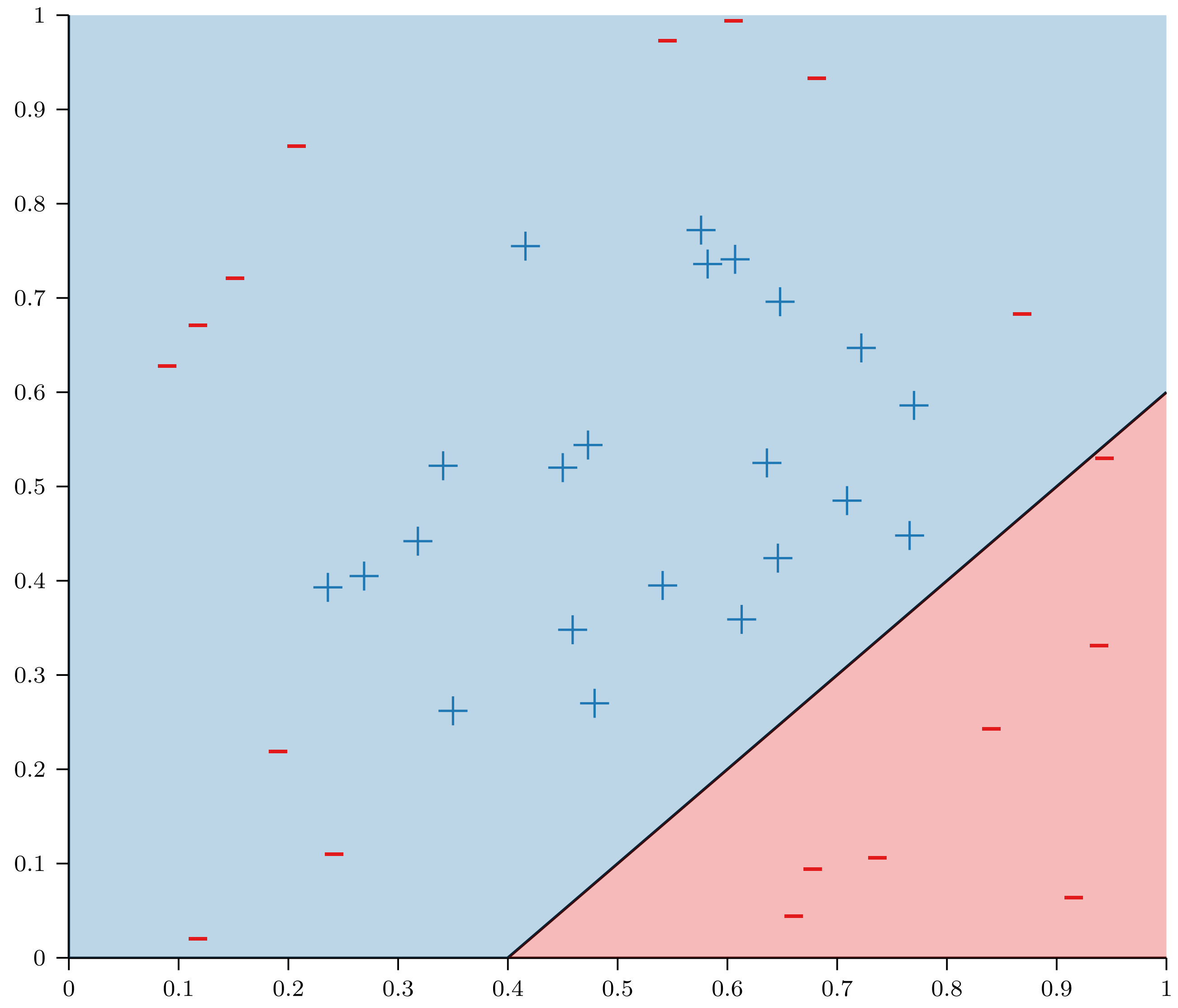
Linear Models



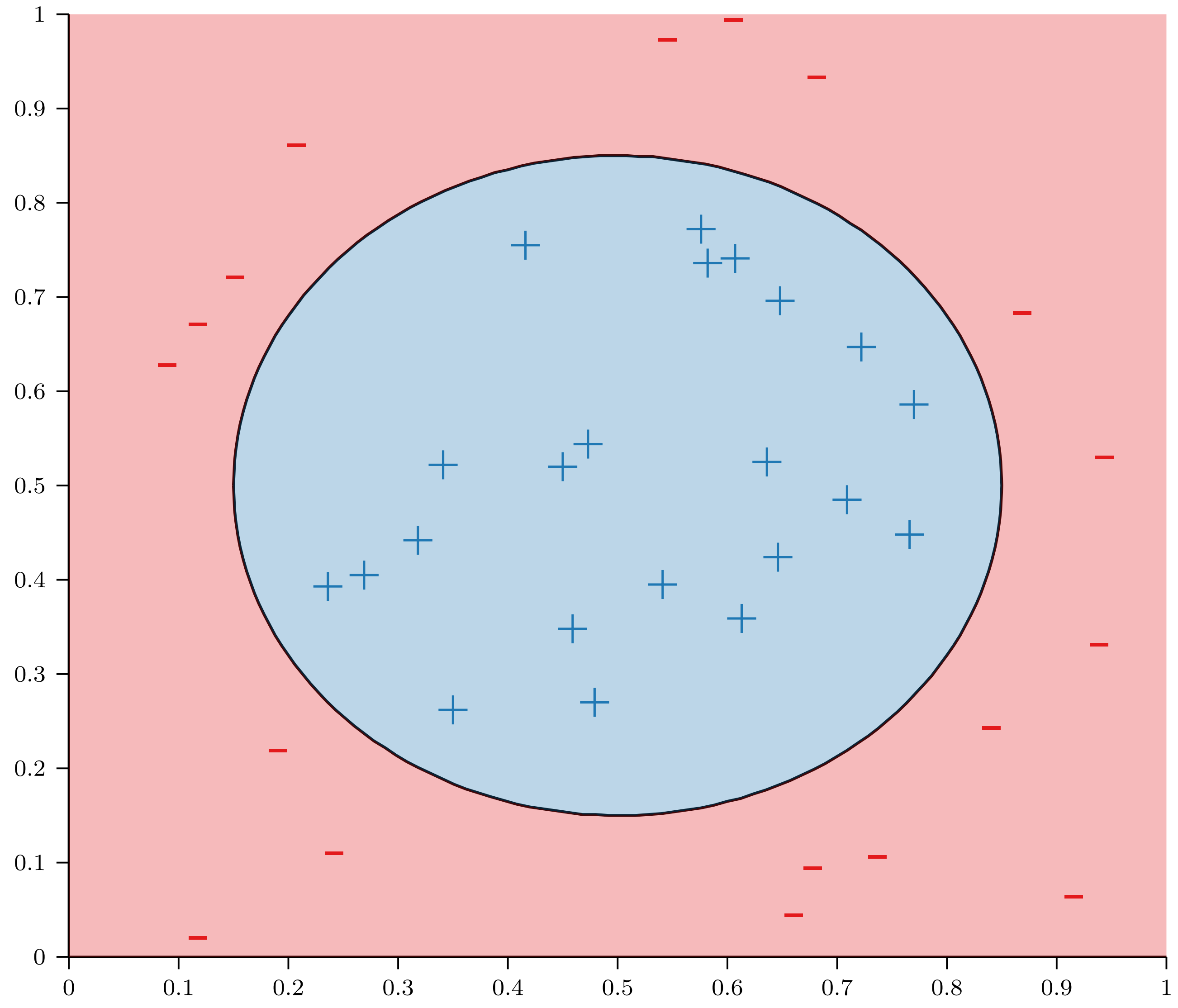
Linear Models?



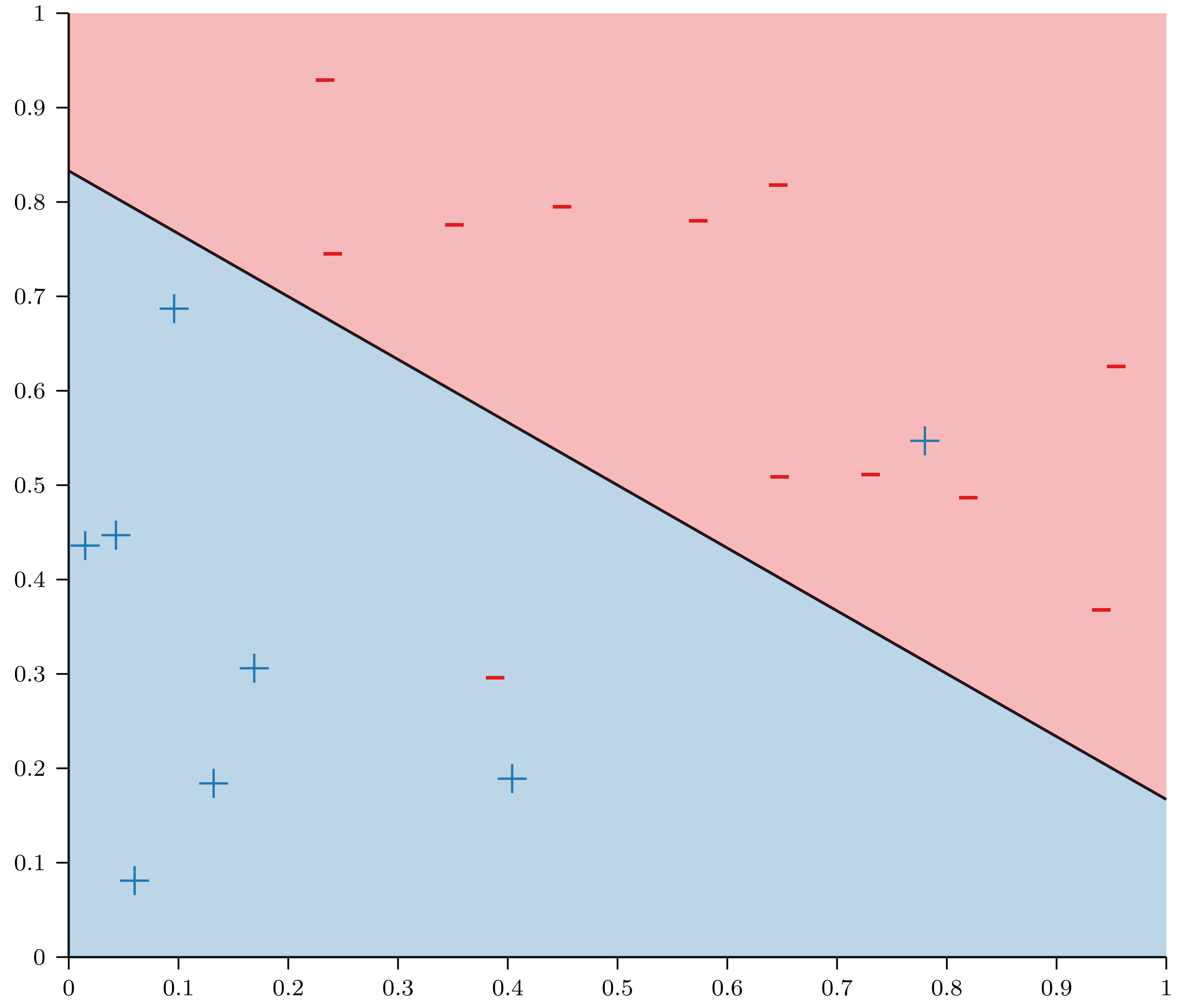
Linear Models?



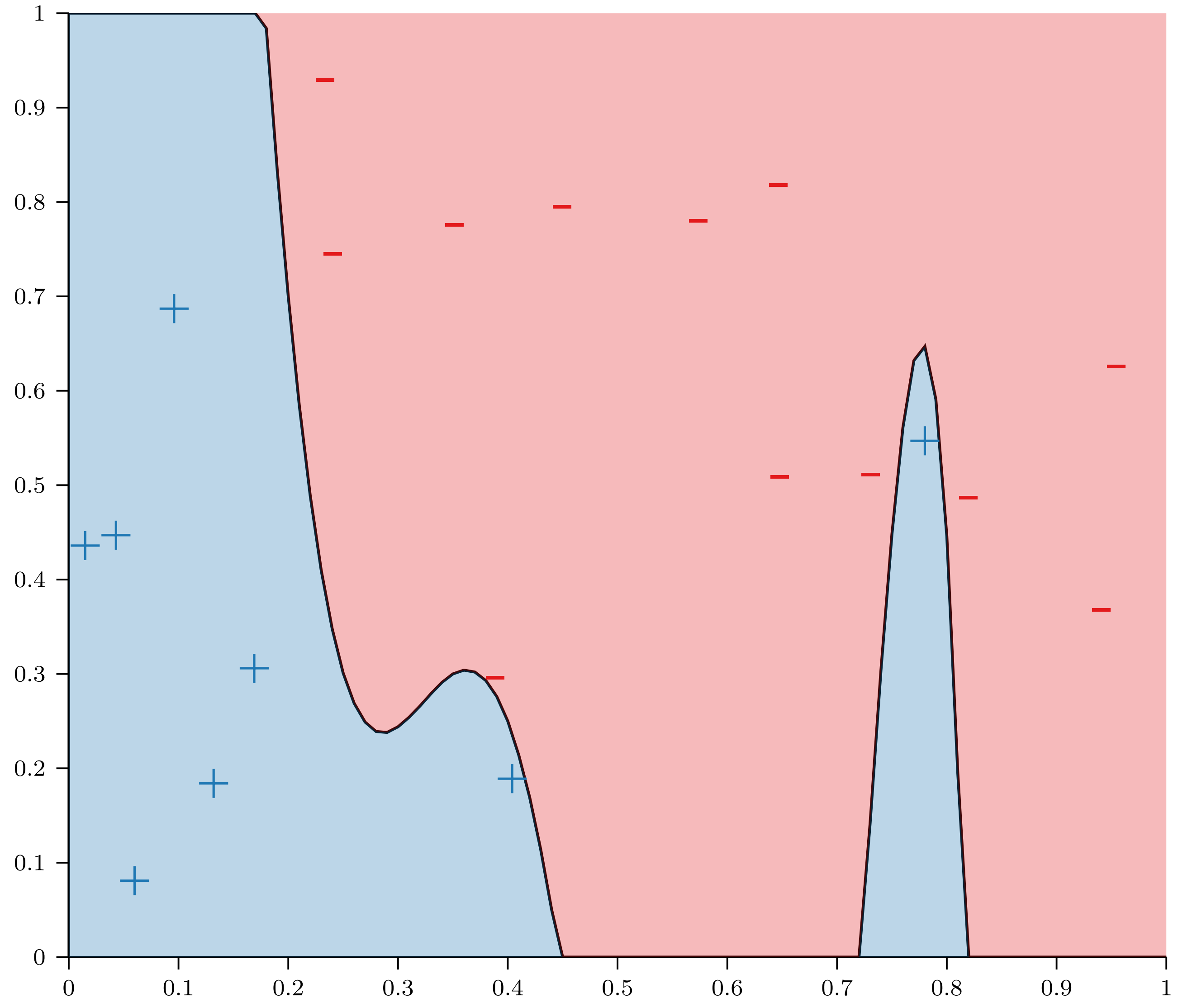
Nonlinear Models



Linear Models



Nonlinear Models?



Feature Transforms: Tradeoffs

	Low-Dimensional Input Space	**High-Dimensional Input Space**
Training Error	High	Low
Generalization	Good	Bad

Overfitting

Regularization

- Constrain models to prevent them from overfitting
- Learning algorithms are optimization problems and regularization imposes constraints on the optimization

Soft Constraints

minimize $\ell_{\mathcal{D}}(\boldsymbol{\omega}) = (\mathrm{X}\boldsymbol{\omega} - \boldsymbol{y})^T(\mathrm{X}\boldsymbol{\omega} - \boldsymbol{y})$

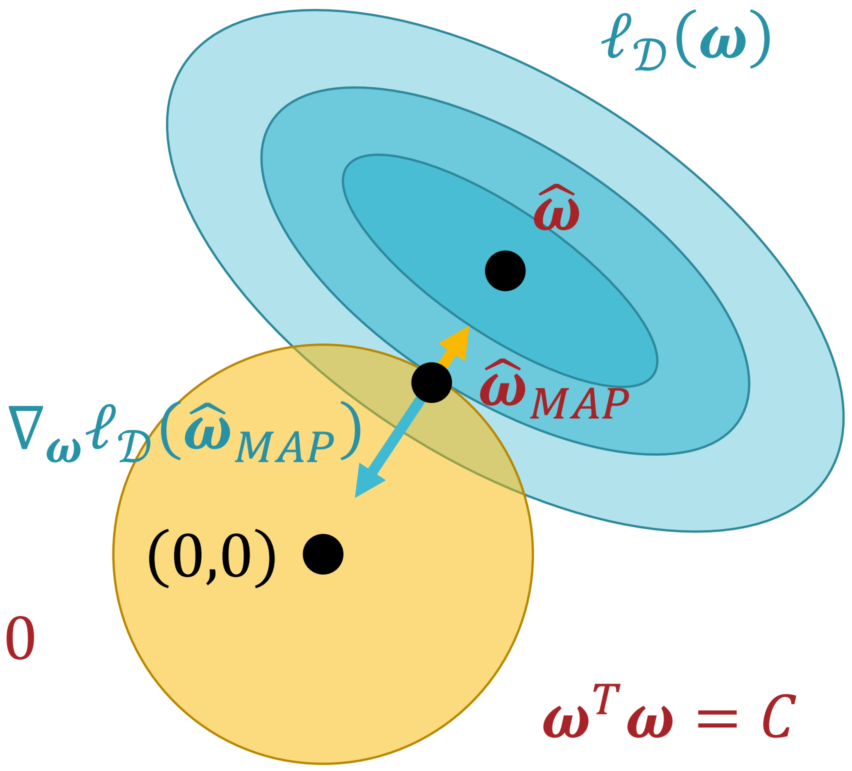
subject to $\boldsymbol{\omega}^T\boldsymbol{\omega} \leq C$

$\nabla_{\omega}\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) \propto -2\widehat{\boldsymbol{\omega}}_{MAP}$

$\nabla_{\omega}\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) = -2\lambda_C\widehat{\boldsymbol{\omega}}_{MAP}$

$\nabla_{\omega}\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) + 2\lambda_C\widehat{\boldsymbol{\omega}}_{MAP} = 0$

$\nabla_{\omega}(\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP}) + \lambda_C(\widehat{\boldsymbol{\omega}}_{MAP})^T\widehat{\boldsymbol{\omega}}_{MAP}) = 0$



$\ell_{\mathcal{D}}(\boldsymbol{\omega})$
$\widehat{\boldsymbol{\omega}}$
$\widehat{\boldsymbol{\omega}}_{MAP}$
$\nabla_{\omega}\ell_{\mathcal{D}}(\widehat{\boldsymbol{\omega}}_{MAP})$
(0,0)
$\boldsymbol{\omega}^T\boldsymbol{\omega} = C$

Soft Constraints: Solving for $\hat{\boldsymbol{\omega}}_{MAP}$

minimize $\ell_{\mathcal{D}}(\boldsymbol{\omega}) = (\mathrm{X}\boldsymbol{\omega} - \boldsymbol{y})^T(\mathrm{X}\boldsymbol{\omega} - \boldsymbol{y})$

subject to $\boldsymbol{\omega}^T\boldsymbol{\omega} \leq C$

$$\Updownarrow$$

minimize $\ell_{\mathcal{D}}^{AUG}(\boldsymbol{\omega}) = \ell_{\mathcal{D}}(\boldsymbol{\omega}) + \lambda_C\boldsymbol{\omega}^T\boldsymbol{\omega}$

Ridge Regression

minimize $\ell_{\mathcal{D}}^{AUG}(\boldsymbol{\omega}) = \ell_{\mathcal{D}}(\boldsymbol{\omega}) + \lambda_C \boldsymbol{\omega}^T \boldsymbol{\omega}$

$$\nabla_{\boldsymbol{\omega}} \ell_{\mathcal{D}}^{AUG}(\boldsymbol{\omega}) = 2(\mathrm{X}^T\mathrm{X}\boldsymbol{\omega} - \mathrm{X}^T\boldsymbol{y} + \lambda_C \boldsymbol{\omega})$$

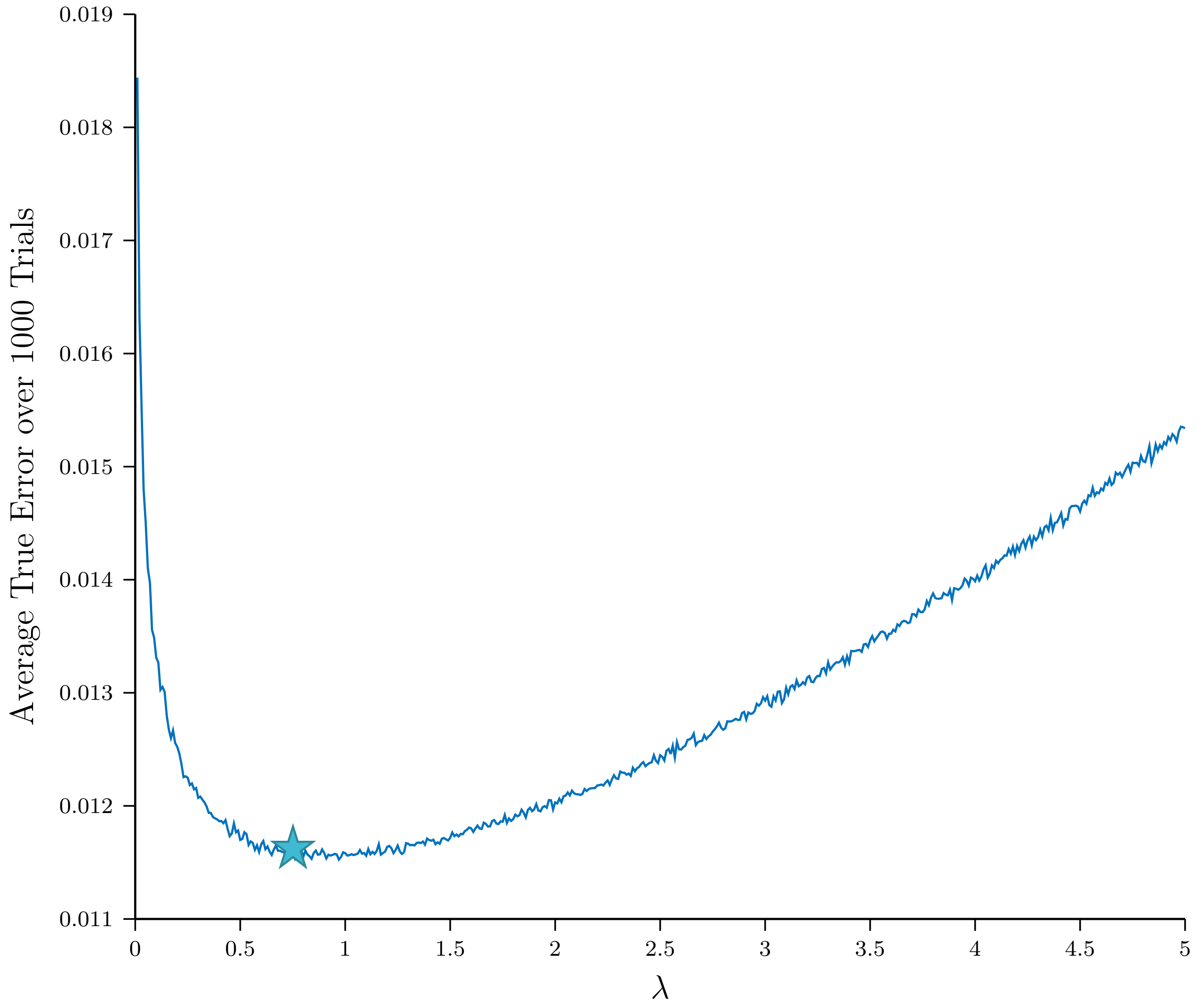
$$2(\mathrm{X}^T\mathrm{X}\hat{\boldsymbol{\omega}}_{MAP} - \mathrm{X}^T\boldsymbol{y} + \lambda_C \hat{\boldsymbol{\omega}}_{MAP}) = 0$$

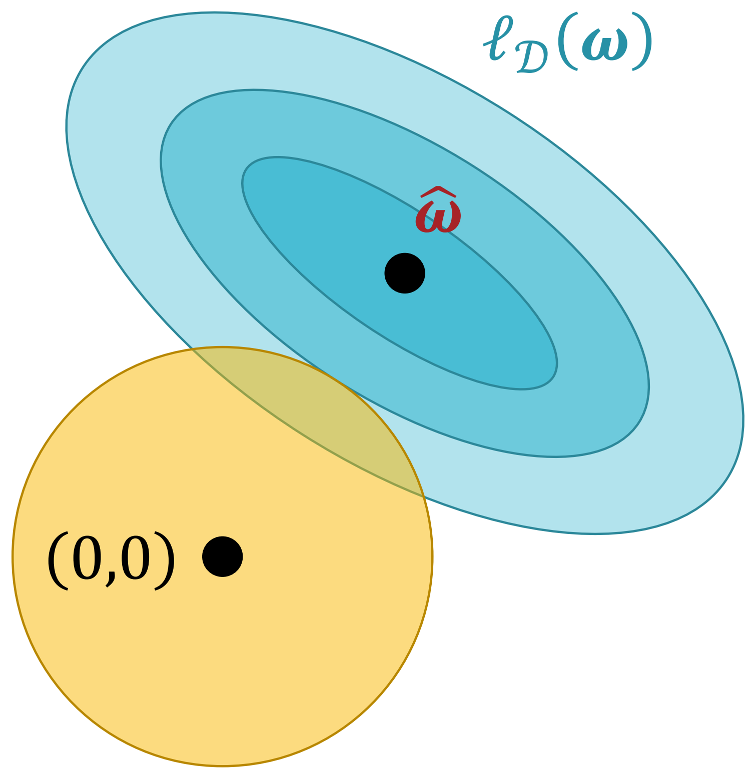
$$(\mathrm{X}^T\mathrm{X} + \lambda_C I_{D+1})\hat{\boldsymbol{\omega}}_{MAP} = \mathrm{X}^T\boldsymbol{y}$$

$$\hat{\boldsymbol{\omega}}_{MAP} = (\mathrm{X}^T\mathrm{X} + \underbrace{\lambda_C I_{D+1}}_{})^{-1}\mathrm{X}^T\boldsymbol{y}$$

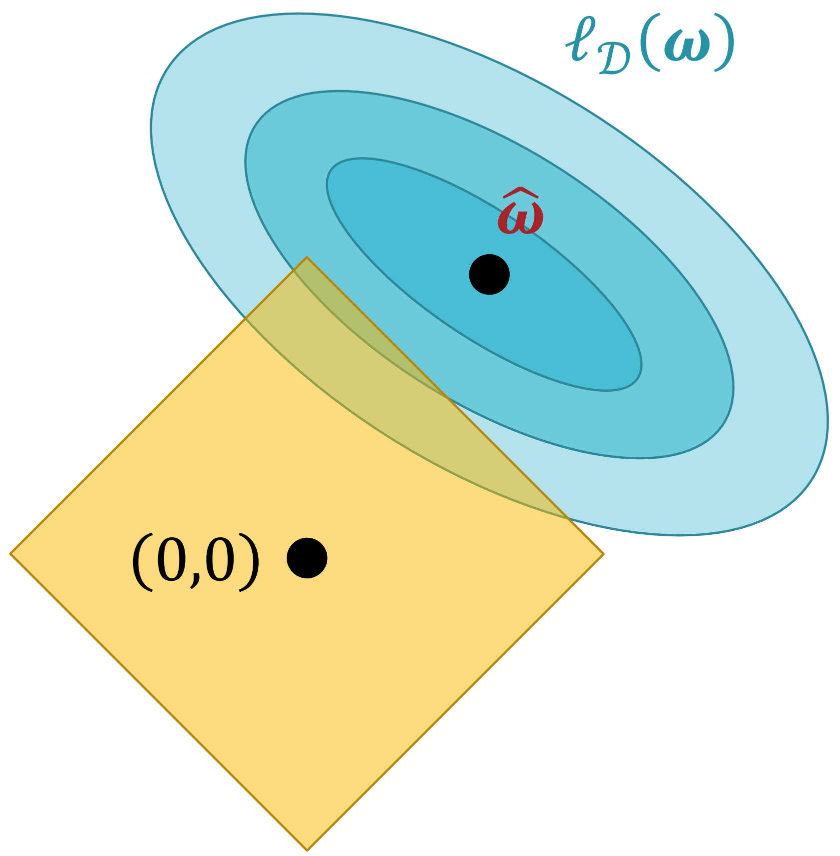
Adding this positive ($\lambda_C \geq 0$) diagonal matrix can help if $\mathrm{X}^T\mathrm{X}$ is not invertible!

Setting λ

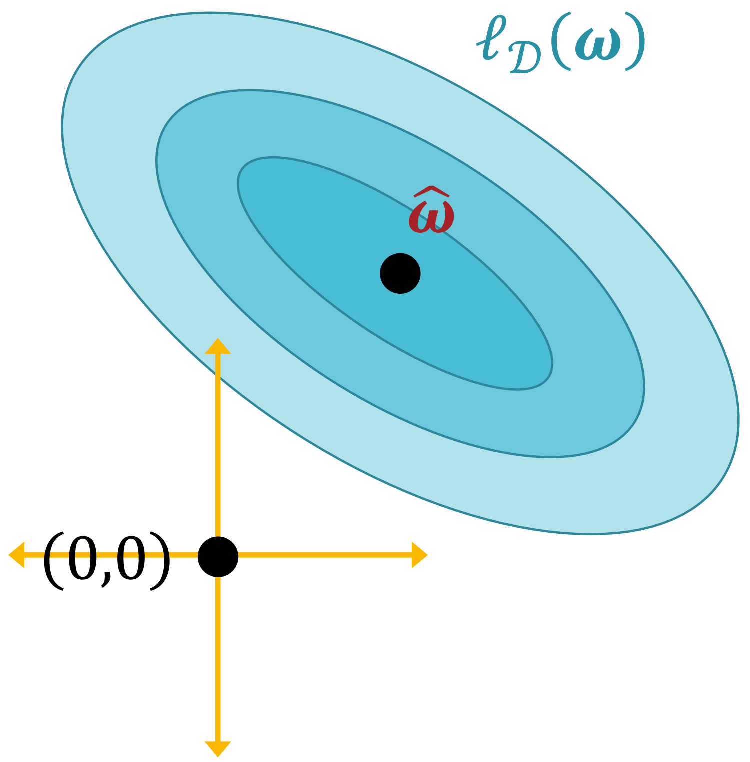



$\ell_{\mathcal{D}}(\boldsymbol{\omega})$
$\hat{\boldsymbol{\omega}}$
$(0,0)$

Ridge or $L2$


$\ell_{\mathcal{D}}(\boldsymbol{\omega})$
$\hat{\boldsymbol{\omega}}$
$(0,0)$

Lasso or $L1$


$\ell_{\mathcal{D}}(\boldsymbol{\omega})$
$\hat{\boldsymbol{\omega}}$
$(0,0)$

$L0$

Other Regularizers

M(C)LE for Linear Regression

- If we assume a linear model with additive Gaussian noise

$$y = \boldsymbol{\omega}^T \boldsymbol{x} + \epsilon \text{ where } \epsilon \sim N(0, \sigma^2) \rightarrow y \sim N(\boldsymbol{\omega}^T \boldsymbol{x}, \sigma^2)$$

- Then given $X = \begin{bmatrix} 1 & \boldsymbol{x}^{(1)} \\ 1 & \boldsymbol{x}^{(2)} \\ \vdots & \vdots \\ 1 & \boldsymbol{x}^{(N)} \end{bmatrix}$ and $\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}$ the MLE of $\boldsymbol{\omega}$ is

$$\widehat{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega}}{\operatorname{argmax}} \log P(\boldsymbol{y} | X, \boldsymbol{\omega})$$

$$= \underset{\boldsymbol{\omega}}{\operatorname{argmax}} \log \exp\left(-\frac{1}{2\sigma^2}(X\boldsymbol{\omega} - \boldsymbol{y})^T (X\boldsymbol{\omega} - \boldsymbol{y})\right)$$

$$= \underset{\boldsymbol{\omega}}{\operatorname{argmin}} (X\boldsymbol{\omega} - \boldsymbol{y})^T (X\boldsymbol{\omega} - \boldsymbol{y}) = (X^T X)^{-1} X^T \boldsymbol{y}$$

MAP for Linear Regression

- If we assume a linear model with additive Gaussian noise

$$y = \boldsymbol{\omega}^T \boldsymbol{x} + \epsilon \text{ where } \epsilon \sim N(0, \sigma^2) \rightarrow y \sim N(\boldsymbol{\omega}^T \boldsymbol{x}, \sigma^2)$$

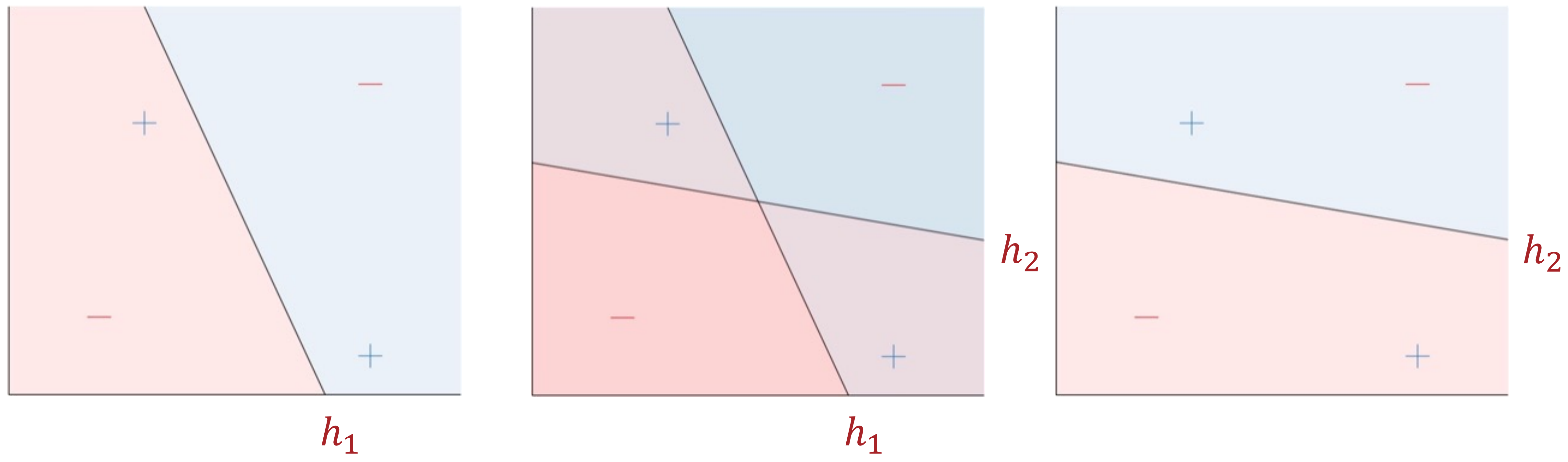
and independent Gaussian priors on all the weights…

$$\omega_d \sim N\left(0, \frac{\sigma^2}{\lambda}\right) \rightarrow p(\boldsymbol{\omega}) \propto \exp\left(-\frac{1}{2\sigma^2}(\lambda \boldsymbol{\omega}^T \boldsymbol{\omega})\right)$$

- … then, the MAP of $\boldsymbol{\omega}$ is the ridge regression solution!

$$\hat{\boldsymbol{\omega}}_{MAP} = \underset{\boldsymbol{\omega}}{\operatorname{argmin}} \, (X\boldsymbol{\omega} - \boldsymbol{y})^T (X\boldsymbol{\omega} - \boldsymbol{y}) + \lambda \boldsymbol{\omega}^T \boldsymbol{\omega}$$

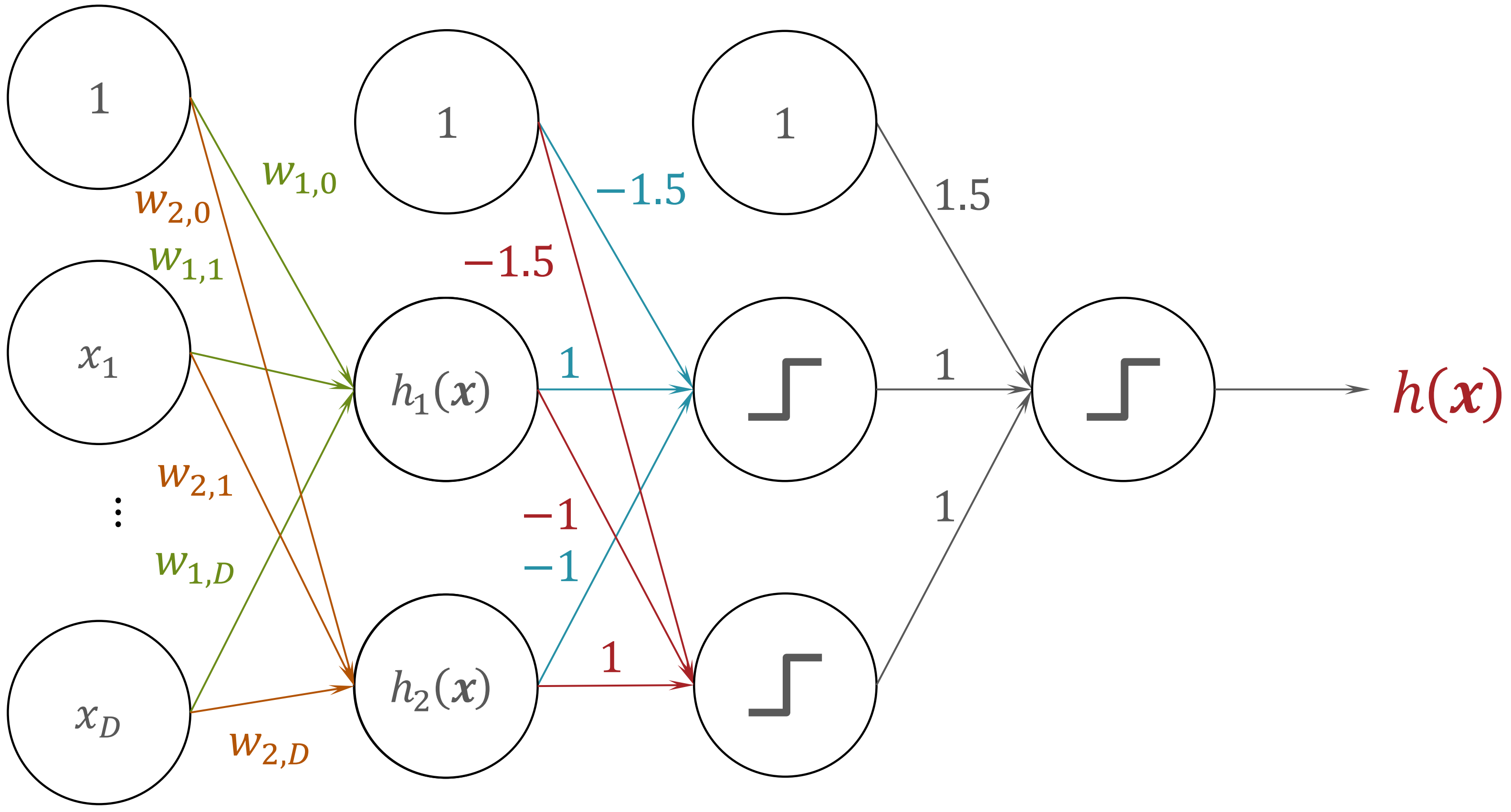
$$= \left(X^{\boldsymbol{T}} X + \lambda I_{D+1}\right)^{-1} X^T \boldsymbol{y}$$



Combining Perceptrons

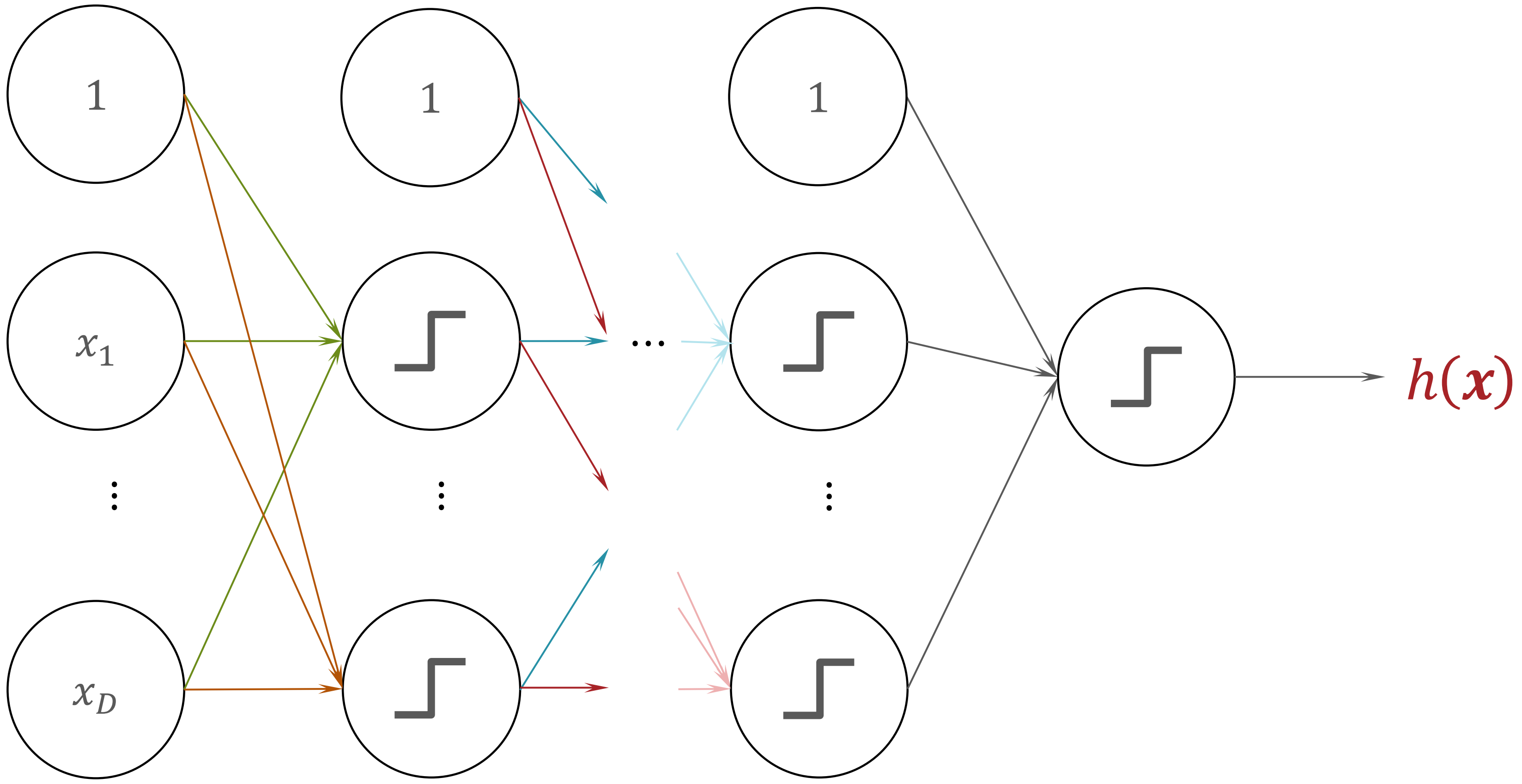
Building a Network

$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$

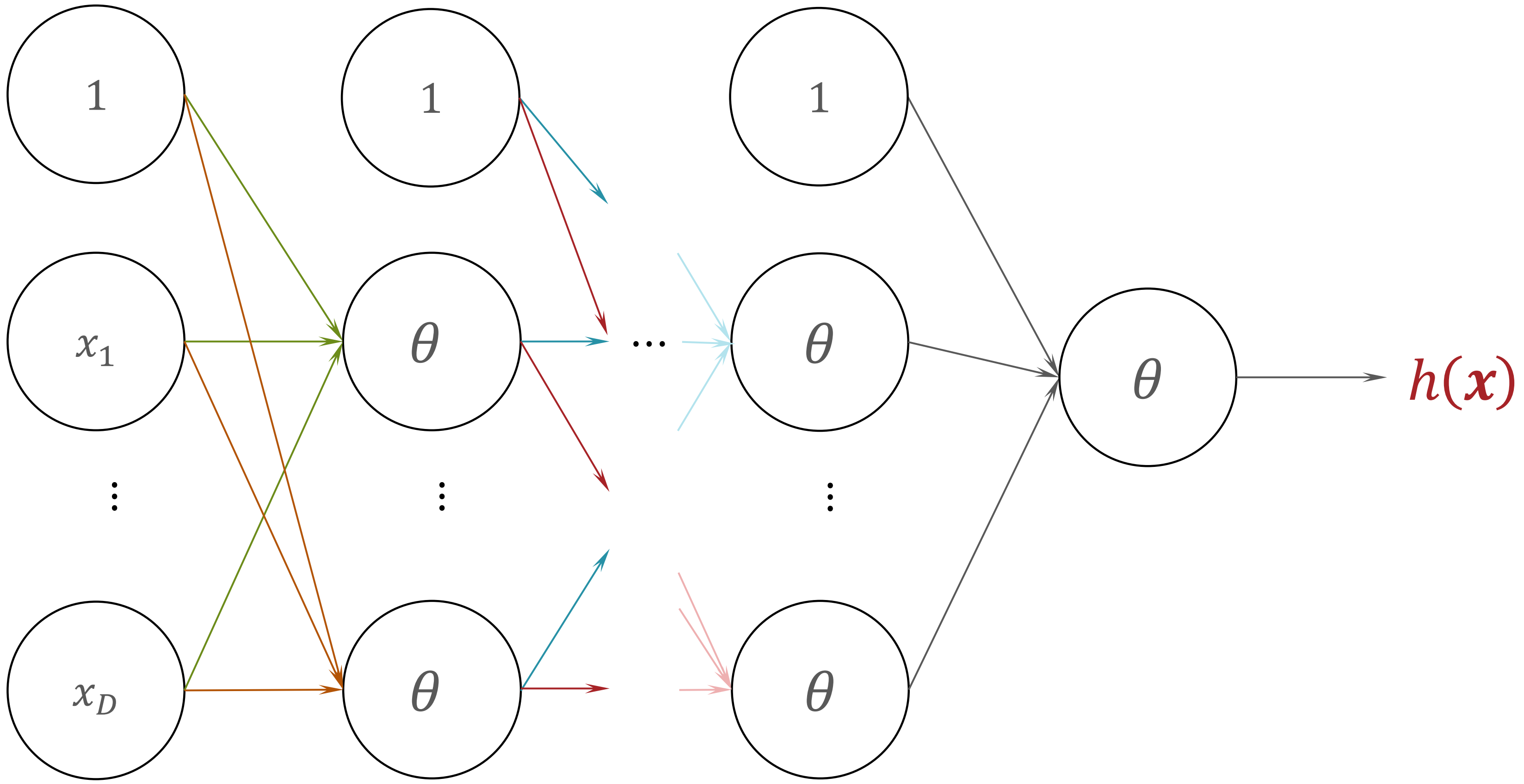


$$h(\boldsymbol{x}) = \text{sign}(\text{sign}(\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) - \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) +$$
$$\text{sign}(-\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) + \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) + 1.5)$$

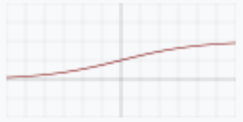
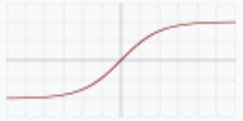
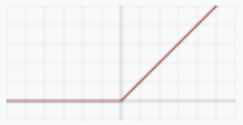
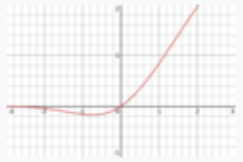
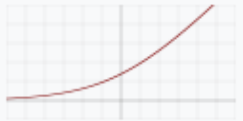
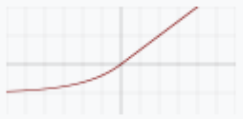
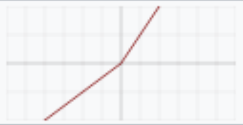
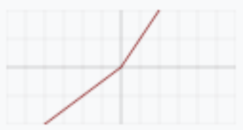
Multi-Layer Perceptron (MLP)



(Fully-Connected) Feed Forward Neural Network



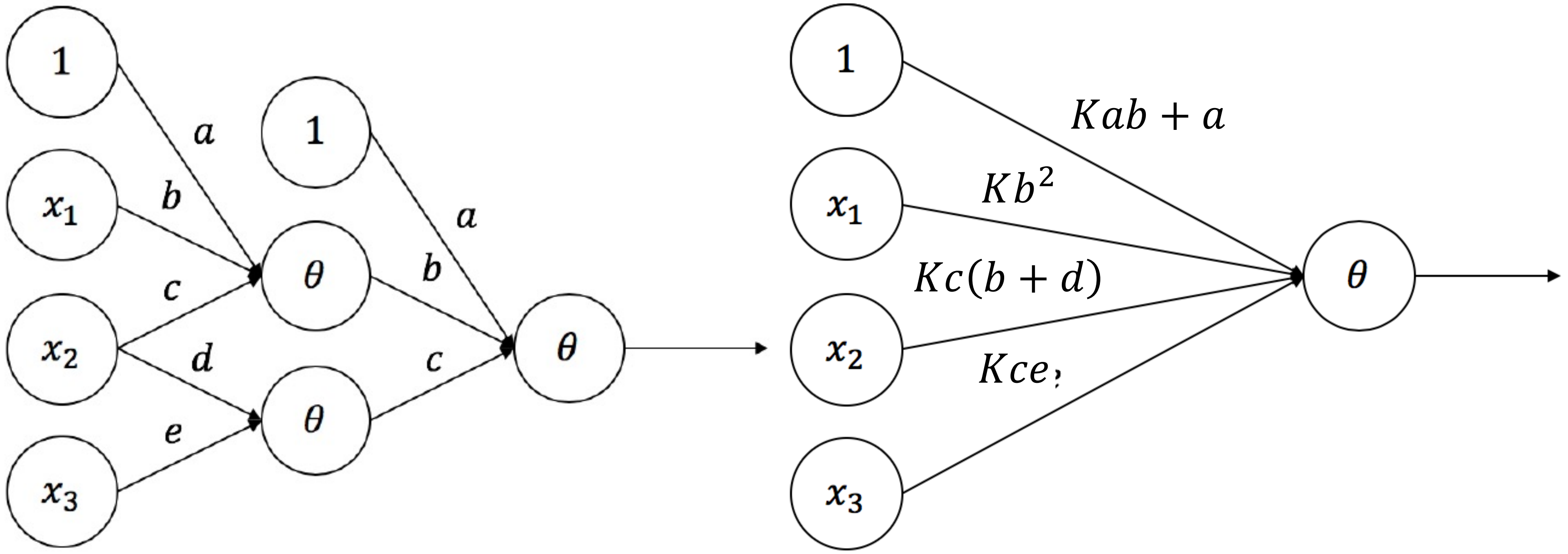
Other Activation Functions

Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$
Softplus[8]		$\ln(1 + e^x)$
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Leaky rectified linear unit (Leaky ReLU)[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \ge 0 \end{cases}$
Parametric rectified linear unit (PReLU)[12]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \ge 0 \end{cases}$ with parameter α

Source: https://en.wikipedia.org/wiki/Activation_function

Practice Problem: Neural Networks

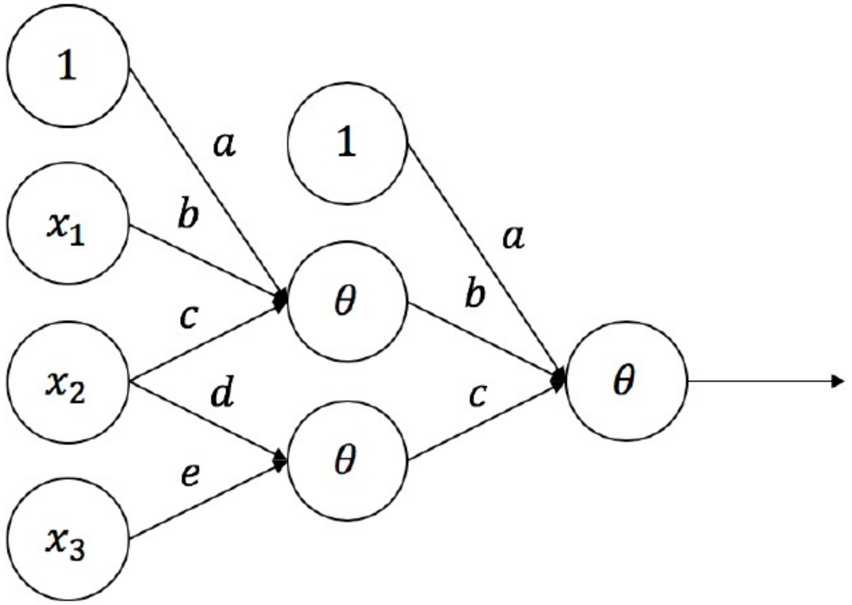
- Consider the following 2-layer neural network:



- Assume we use a linear activation function $\theta(a) = Ka$ for some constant K. Draw a 1-layer neural network with the same input and output layers that is equivalent to the one above

Practice Problem: Neural Networks

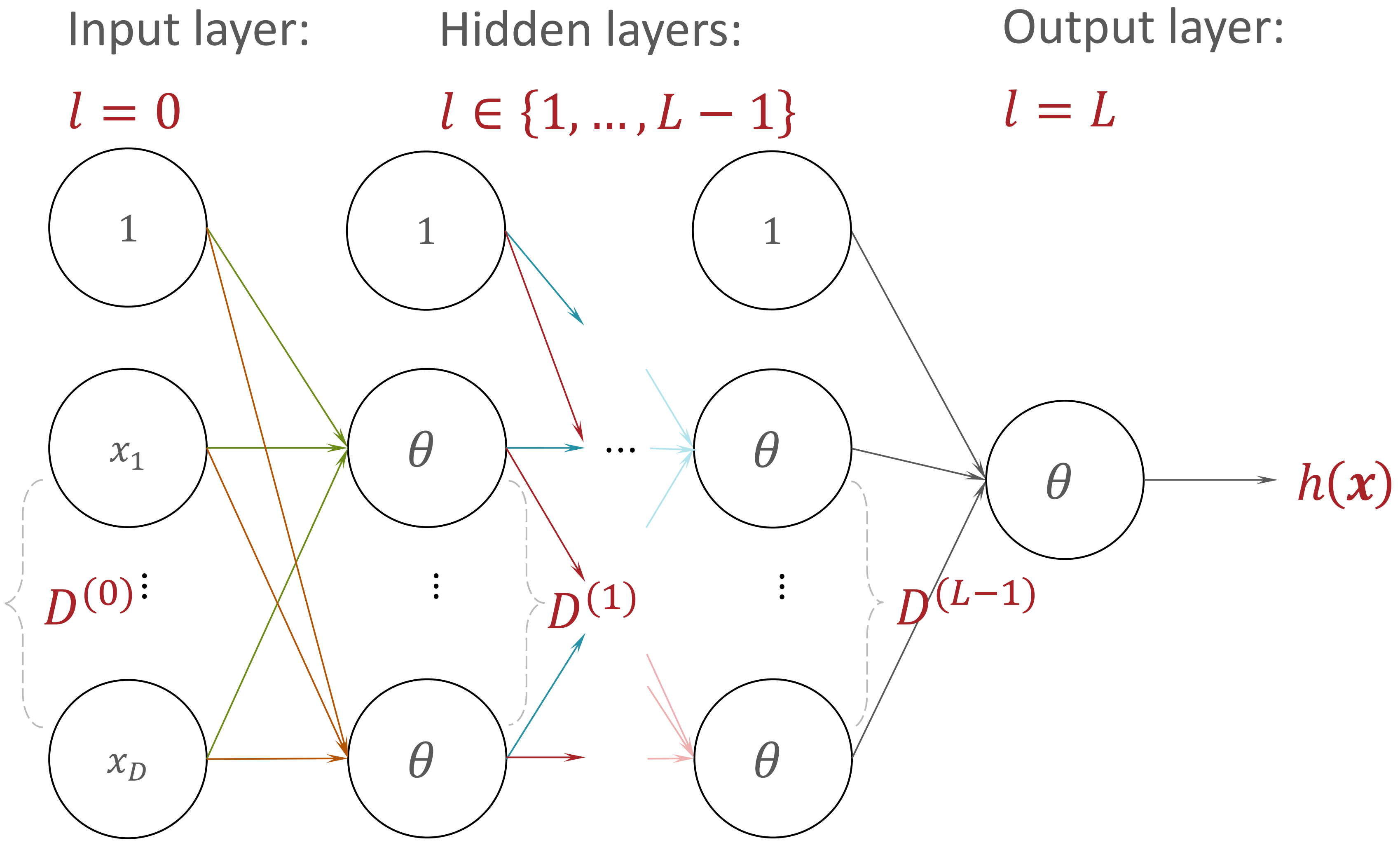
- Consider the following 2-layer neural network:



- Now assume we use the tanh activation function. Can you still draw a 1-layer neural network that is equivalent to the one above? If so, draw it and if not, briefly justify your answer.

- No, there is no way to recreate the nested tanh functions using just one weighted sum

(Fully-Connected) Feed Forward Neural Network

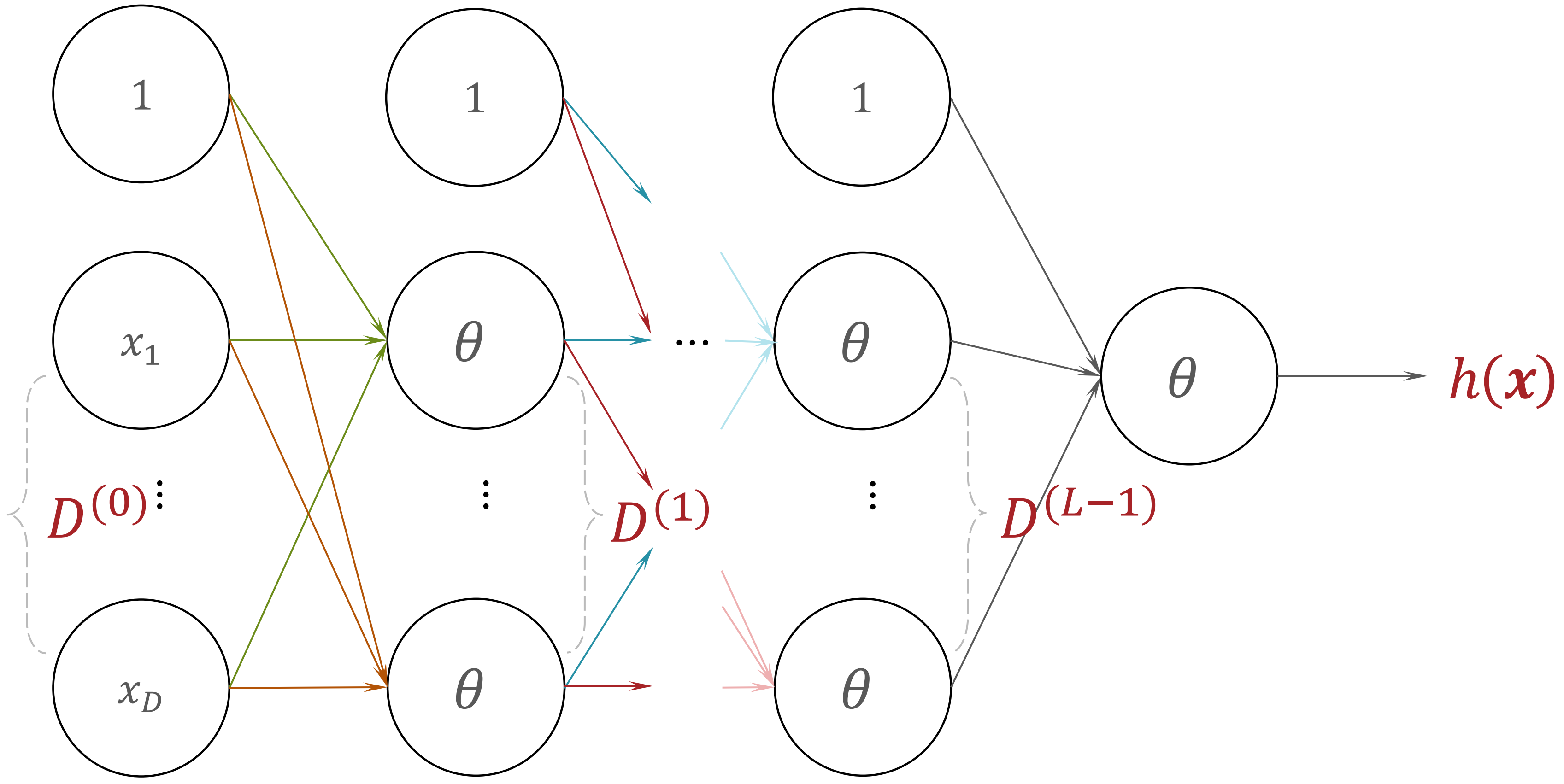


Layer l has dimension $D^{(l)} \rightarrow$ Layer l has $D^{(l)} + 1$ nodes, counting the bias node

(Fully-Connected) Feed Forward Neural Network

The weights between layer $l-1$ and layer l are a matrix:

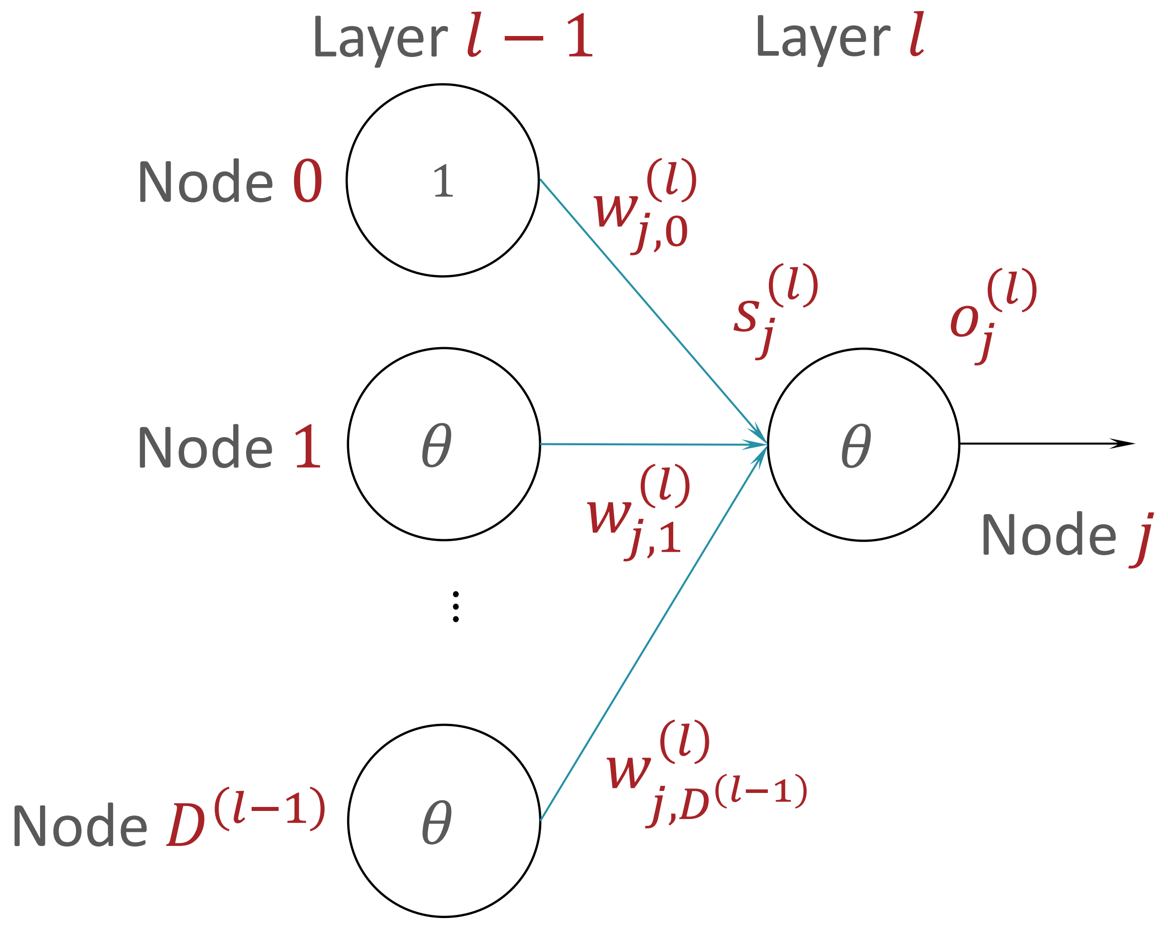
$W^{(l)} \in \mathbb{R}^{D^{(l)} \times (D^{(l-1)}+1)}$



$w_{j,i}^{(l)}$ is the weight between node i in layer $l-1$ and node j in layer l

Signal and Outputs

Every node has an incoming *signal* and outgoing *output*



$$\boldsymbol{s}^{(l)} = W^{(l)}\boldsymbol{o}^{(l-1)} \text{ and } \boldsymbol{o}^{(l)} = \left[1, \theta\left(\boldsymbol{s}^{(l)}\right)\right]^T$$

Forward Propagation for Making Predictions

- Input: weights $W^{(1)}, \ldots, W^{(L)}$ and a query data point $\boldsymbol{x}$
- Initialize $\boldsymbol{o}^{(0)} = [1, \boldsymbol{x}]^T$
- For $l = 1, \ldots, L$
  - $\boldsymbol{s}^{(l)} = W^{(l)} \boldsymbol{o}^{(l-1)}$
  - $\boldsymbol{o}^{(l)} = \left[1, \theta\left(\boldsymbol{s}^{(l)}\right)\right]^T$
- Output: $h_{W^{(1)}, \ldots, W^{(L)}}(\boldsymbol{x}) = \boldsymbol{o}^{(L)}$

Gradient Descent for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta^{(0)}$
- Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$ (???)
- While TERMINATION CRITERION is not satisfied (???)
  - For $l = 1, \ldots, L$
    - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)$ (???)
    - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
  - Increment t: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$

Computing Gradients: Intuition

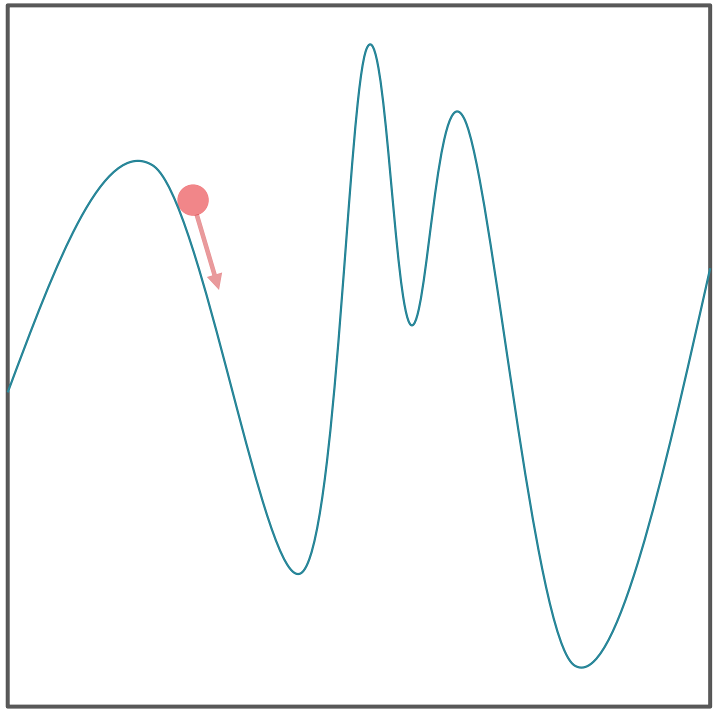
- A weight affects the prediction of the network (and therefore the error) through downstream signals/outputs
    - Use the chain rule!
- Any weight going into the same node will affect the prediction through the same downstream path
    - Compute derivatives starting from the last layer and move “backwards”
    - Store computed derivatives and reuse for efficiency (dynamic programming)

Back-propagation

- Input: $W^{(1)}, \dots, W^{(L)}$ and $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
- Initialize: $\ell_{\mathcal{D}} = 0$ and $G^{(l)} = 0 \odot W^{(l)} \; \forall \, l = 1, \dots, L$
- For $n = 1, \dots, N$
  - Run forward propagation with $\boldsymbol{x}^{(n)}$ to get $\boldsymbol{o}^{(1)}, \dots, \boldsymbol{o}^{(L)}$
  - (Optional) Increment $\ell_{\mathcal{D}}$: $\ell_{\mathcal{D}} = \ell_{\mathcal{D}} + \left(o^{(L)} - y^{(n)}\right)^2$
  - Initialize: $\boldsymbol{\delta}^{(L)} = 2\left(o_1^{(L)} - y^{(n)}\right)\left(1 - \left(o_1^{(L)}\right)^2\right)$
  - For $l = L - 1, \dots, 1$
    - Compute $\boldsymbol{\delta}^{(l)} = W^{(l+1)^T} \boldsymbol{\delta}^{(l+1)} \odot \left(1 - \boldsymbol{o}^{(l)} \odot \boldsymbol{o}^{(l)}\right)$
    - Increment $G^{(l)}$: $G^{(l)} = G^{(l)} + \boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T}$
- Output: $G^{(1)}, \dots, G^{(L)}$, the gradients of $\ell_{\mathcal{D}}$ w.r.t $W^{(1)}, \dots, W^{(L)}$

Non-convexity

- Gradient descent is not guaranteed to find a global minimum on non-convex surfaces



Stochastic Gradient Descent for Neural Networks

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{SGD}^{(0)}$

1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
    a. Randomly sample a data point from $\mathcal{D}$, $\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)$
    b. Compute the pointwise gradient,

    $$G^{(l)} = \nabla_{W^{(l)}} e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right) \forall\, l$$

    c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_{t}^{(l)} - \eta_{SGD}^{(0)} G^{(l)} \forall\, l$
    d. Increment t: $t \leftarrow t + 1$

- Output: $W_{t}^{(1)}, \ldots, W_{t}^{(L)}$

Mini-batch Stochastic Gradient Descent for Neural Networks

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$

1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,
   $$G^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} e\left(\boldsymbol{o}^{(L)}, y^{(b)}\right) \ \forall\, l$$
   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta_{MB}^{(0)} G^{(l)} \ \forall\, l$
   d. Increment t: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \ldots, W_t^{(L)}$

Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B, \beta$

1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $G_{-1}^{(l)} = 0 \odot W^{(l)} \ \forall\, l = 1, \dots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,
   $$G_t^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} e\left(\boldsymbol{o}^{(L)}, y^{(b)}\right) \forall\, l$$
   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta_{MB}^{(0)}\left(\beta G_{t-1}^{(l)} + G_t^{(l)}\right) \forall\, l$
   d. Increment t: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \dots, W_t^{(L)}$

Mini-batch Stochastic Gradient Descent with Adaptive Gradients for Neural Networks

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B, \epsilon$

1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0, S_{-1}^{(l)} = 0 \odot W^{(l)} \; \forall \, l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,
   $$G_t^{(l)} = \frac{1}{B} \sum_{b=1}^{B} \nabla_{W^{(l)}} e\left(\boldsymbol{o}^{(L)}, y^{(b)}\right) \; \forall \, l$$
   c. Update $S^{(l)}$: $S_t^{(l)} = S_{t-1}^{(l)} + G_t^{(l)} \odot G_t^{(l)} \; \forall \, l$
   d. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \frac{\eta_{MB}^{(0)}}{\sqrt{S_t^{(l)} + \epsilon}} \odot G_t^{(l)} \; \forall \, l$
   e. Increment t: $t \leftarrow t + 1$

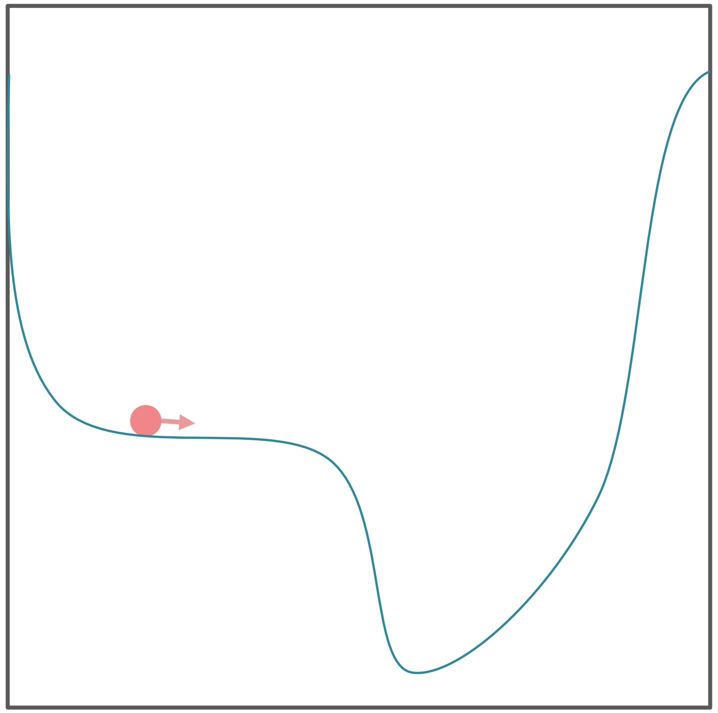
- Output: $W_t^{(1)}, \ldots, W_t^{(L)}$

Random Restarts

- Run mini-batch gradient descent (with momentum & adaptive gradients) multiple times, each time starting with a ***different***, ***random*** initialization for the weights.
- Compute the training error of each run at termination and return the set of weights that achieves the lowest training error.

Terminating Gradient Descent

- For non-convex surfaces, the gradient’s magnitude is often not a good metric for proximity to a minimum



Terminating Gradient Descent "Early"

- For non-convex surfaces, the gradient's magnitude is often not a good metric for proximity to a minimum
- Combine multiple termination criteria e.g. only stop if enough iterations have passed and the improvement in error is small
- Alternatively, terminate early by using a validation data set: if the validation error starts to increase, just stop!
    - Early stopping asks like regularization by **limiting how much of the hypothesis set** is explored

Neural Networks and Regularization

- Minimize $\ell_{\mathcal{D}}^{AUG}\left(W^{(1)}, \dots, W^{(L)}, \lambda_C\right)$

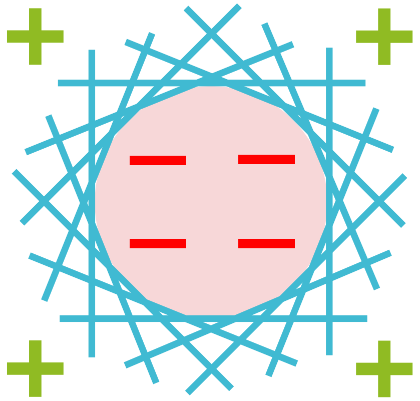
$$= \ell_{\mathcal{D}}\left(W^{(1)}, \dots, W^{(L)}\right) + \lambda_C \Omega\left(W^{(1)}, \dots, W^{(L)}\right)$$

e.g. L2 regularization

$$\Omega\left(W^{(1)}, \dots, W^{(L)}\right) = \sum_{l=1}^{L} \sum_{i=0}^{d^{(l-1)}} \sum_{j=1}^{d^{(l)}} \left(w_{j,i}^{(l)}\right)^2$$

MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



- Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 3-layer MLP

NNs as Universal Approximators (Cybenko, 1989 & Hornik, 1991)

- Theorem: Any bounded, continuous function can be approximated to an arbitrary precision using a 2-layer (1 hidden layer) feed-forward NN if the activation function, θ, is continuous, bounded and non-constant.
- What about unbounded or discontinuous functions?
- Use more layers!

Source: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F21A09B7475DFB9487990020839A39D2?doi=10.1.1.441.7873&rep=rep1&type=pdf
https://doi.org/10.1016/0893-6080%2891%2990009-T

NNs as Universal Approximators (Cybenko, 1988)

- Theorem: Any function can be approximated to an arbitrary precision using a 3-layer (2 hidden layers) feed-forward NN if the activation function, θ, is continuous, bounded and non-constant.

Source: G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Dept. of Computer Science, Tufts University, Medford, MA, 1988.

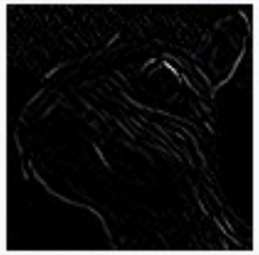
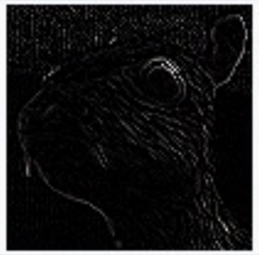
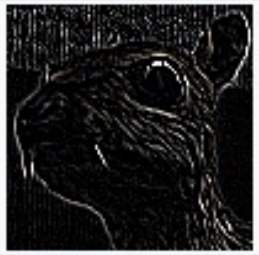
Practice Problem: Deep Learning

- Consider two models: 1) a deep neural network with nonlinear activation functions and 2) logistic regression with some fixed nonlinear feature transformation. Briefly describe one scenario where you would prefer model 2) to model 1).
- If you don't have enough training data, using a deep neural network to learn the nonlinear features could result in overfitting so you should prefer using fixed nonlinear features. Conversely, given enough training data, learning the nonlinear features could be better.

Convolutional Neural Networks

- Neural networks are frequently applied to inputs with some inherent spatial structure, e.g., images
- Idea: use the first few layers to identify relevant macro-features, e.g., edges
- Insight: for spatially-structured inputs, many useful macro-features are shift or location-invariant, e.g., an edge in the upper left corner of a picture looks like an edge in the center
- Strategy: learn a filter for macro-feature detection in a small window and apply it over the entire image

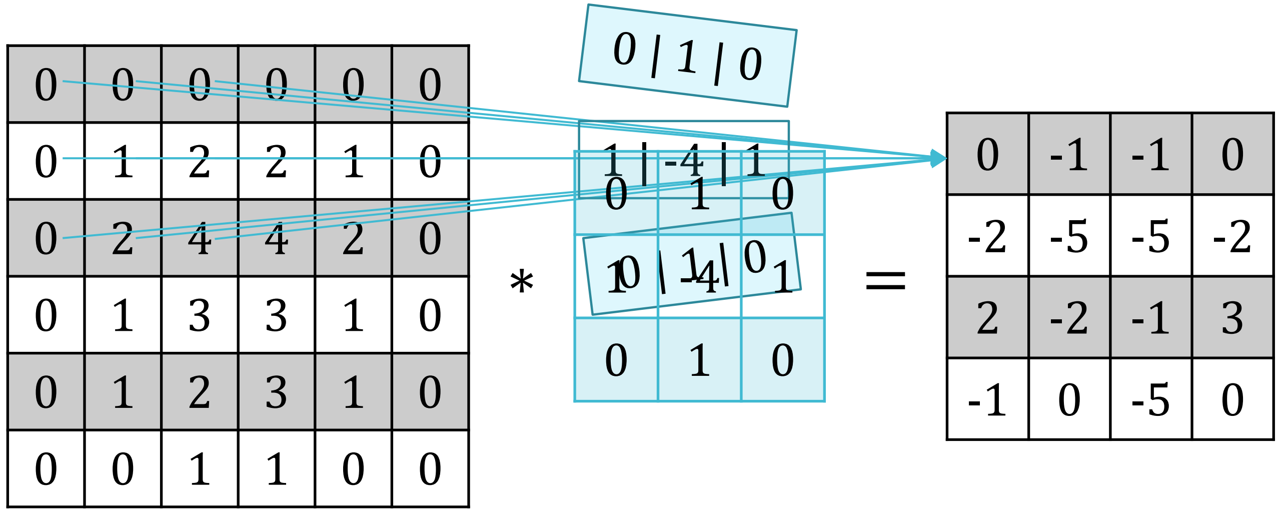
Convolutional Filters

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

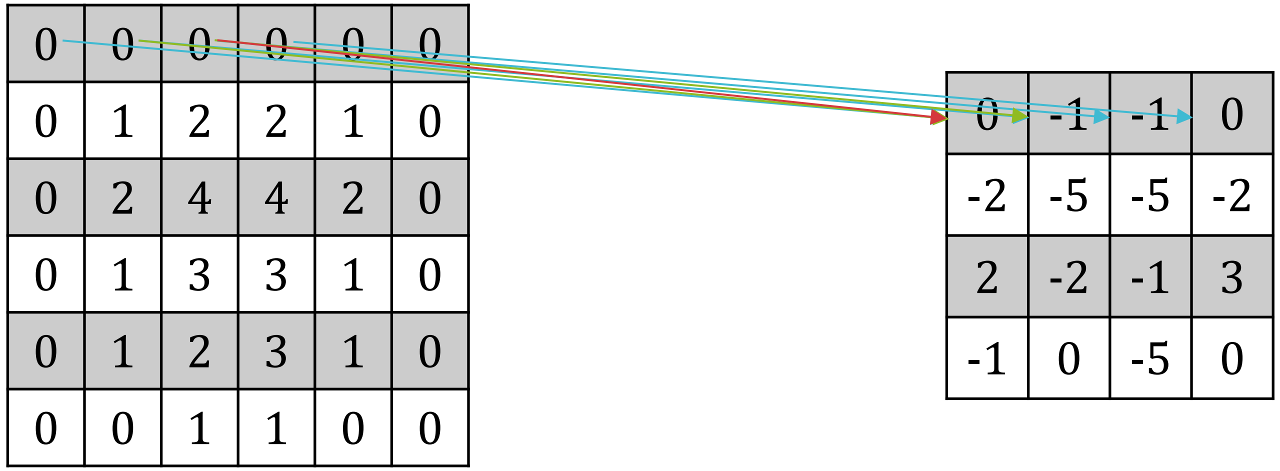
Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 & 0 \\ -2 & -5 & -5 & -2 \\ 2 & -2 & -1 & 3 \\ -1 & 0 & -5 & 0 \end{bmatrix}$$

Convolutional Filters

- Convolutions can be represented by a feed forward neural network where:
  1. Nodes in the input layer are only connected to some nodes in the next layer but not all nodes.
  2. Many of the weights have the same value.



0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

- Many fewer weights than a fully connected layer!
- Convolution weights are learned using gradient descent/ backpropagation, not prespecified

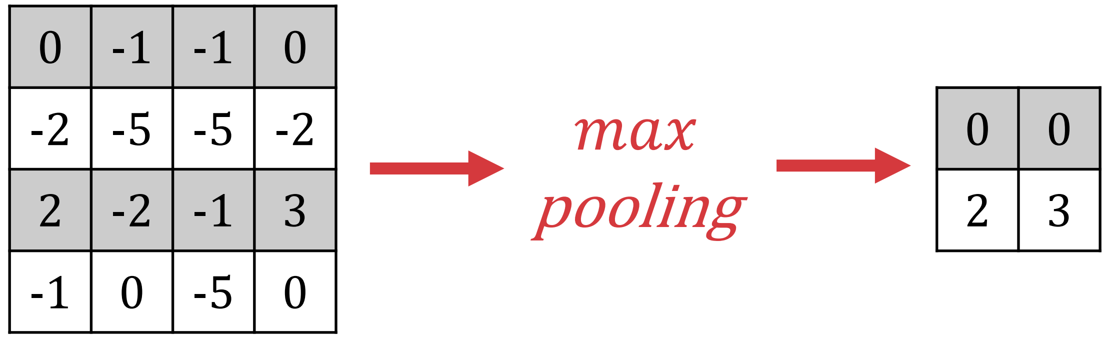
Convolutional Filters: Padding

- What if relevant features exist at the border of our image?
- Add zeros around the image to allow for the filter to be applied "everywhere" e.g. a *padding* of 1 with a 3x3 filter preserves image size and allows every pixel to be the center

$$
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 \\
\hline
0 & 0 & 2 & 4 & 4 & 2 & 0 & 0 \\
\hline
0 & 0 & 1 & 3 & 3 & 1 & 0 & 0 \\
\hline
0 & 0 & 1 & 2 & 3 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
0 & 1 & 0 \\
\hline
1 & -4 & 1 \\
\hline
0 & 1 & 0 \\
\hline
\end{array}
=
\begin{array}{|c|c|c|c|c|c|}
\hline
0 & 1 & 2 & 2 & 1 & 0 \\
\hline
1 & 0 & -1 & -1 & 0 & 1 \\
\hline
2 & -2 & -5 & -5 & -2 & 2 \\
\hline
1 & 2 & -2 & -1 & 3 & 1 \\
\hline
1 & -1 & 0 & -5 & 0 & 1 \\
\hline
0 & 2 & -1 & 0 & 2 & 0 \\
\hline
\end{array}
$$

Downsampling: Pooling

- Combine multiple adjacent nodes into a single node



0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

0	0
2	3

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
  - Protects the network from (slightly) noisy inputs

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 1 \\ 0 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned

- Many relevant macro-features will tend to span large portions of the image, so taking strides with the convolution tends not to miss out on too much

Key Question

- Given a hypothesis with zero/low training error, what can we say about its true error?

PAC Learning

- PAC = **P**robably **A**pproximately **C**orrect

- PAC Criterion:

$$P\left(\left|R(h) - \hat{R}(h)\right| \leq \epsilon\right) \geq 1 - \delta \; \forall \, h \in \mathcal{H}$$

for some ϵ (difference between expected and empirical risk) and δ (probability of “failure”)

  - We want the PAC criterion to be satisfied for $\mathcal{H}$ with small values of ϵ and δ

Sample Complexity

- The sample complexity of an algorithm/hypothesis set is the number of labelled training data points needed to satisfy the PAC criterion for some δ and ϵ
- Four cases
  - Realizable vs. Agnostic
    - Realizable $\rightarrow$ $c^* \in \mathcal{H}$
    - Agnostic $\rightarrow$ c^* might or might not be in $\mathcal{H}$
  - Finite vs. Infinite
    - Finite $\rightarrow$ $|\mathcal{H}| < \infty$
    - Infinite $\rightarrow$ $|\mathcal{H}| = \infty$

Theorem 1: Finite, Realizable Case

- For a finite hypothesis set $\mathcal{H}$ s.t. $c^* \in \mathcal{H}$ and arbitrary distribution p^*, if the number of labelled training data points satisfies

$$M \geq \frac{1}{\epsilon}\left(\ln(|\mathcal{H}|) + \ln\left(\frac{1}{\delta}\right)\right)$$

then with probability at least $1 - \delta$, all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have $R(h) \leq \epsilon$

Statistical Learning Theory Corollary

- For a finite hypothesis set $\mathcal{H}$ s.t. $c^* \in \mathcal{H}$ and arbitrary distribution p^*, given a training data set S s.t. $|S| = M$, all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have

$$R(h) \leq \frac{1}{M}\left(\ln(|\mathcal{H}|) + \ln\left(\frac{1}{\delta}\right)\right)$$

with probability at least $1 - \delta$.

Theorem 2: Finite, Agnostic Case

- For a finite hypothesis set $\mathcal{H}$ and arbitrary distribution p^*, if the number of labelled training data points satisfies

$$M \geq \frac{1}{2\epsilon^2}\left(\ln(|\mathcal{H}|) + \ln\left(\frac{2}{\delta}\right)\right)$$

then with probability at least $1 - \delta$, all $h \in \mathcal{H}$ satisfy $\left|R(h) - \hat{R}(h)\right| \leq \epsilon$

- Bound is inversely quadratic in ϵ, e.g., halving ϵ means we need four times as many labelled training data points
- Solving for ϵ gives…

Statistical Learning Theory Corollary

- For a finite hypothesis set $\mathcal{H}$ and arbitrary distribution p^*, given a training data set S s.t. $|S| = M$, all $h \in \mathcal{H}$ have

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{1}{2M}\left(\ln(|\mathcal{H}|) + \ln\left(\frac{2}{\delta}\right)\right)}$$

with probability at least $1 - \delta$.

Labellings

- Given some finite set of data points $S = \left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(M)}\right)$ and some hypothesis $h \in \mathcal{H}$, applying h to each point in S results in a **labelling**
  - $\left(h\left(\boldsymbol{x}^{(1)}\right), \dots, h\left(\boldsymbol{x}^{(M)}\right)\right)$ is a vector of M +1's and -1's
- Given $S = \left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(M)}\right)$, each hypothesis in $\mathcal{H}$ induces a labelling but not necessarily a unique labelling
  - The set of labellings induced by $\mathcal{H}$ on S is

$$\mathcal{H}(S) = \left\{\left(h\left(\boldsymbol{x}^{(1)}\right), \dots, h\left(\boldsymbol{x}^{(M)}\right)\right) \middle| \, h \in \mathcal{H}\right\}$$

Growth Function

- The **growth function** of $\mathcal{H}$ is the maximum number of distinct labellings $\mathcal{H}$ can induce on ***any*** set of M data points:

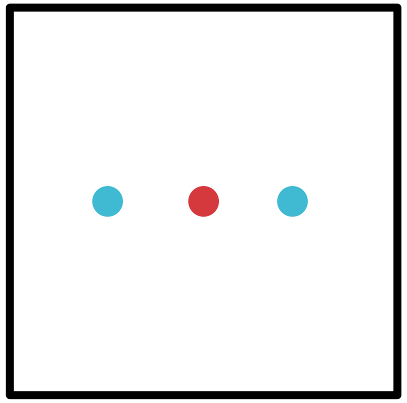
$$g_{\mathcal{H}}(M) = \max_{S \,:\, |S|=M} |\mathcal{H}(S)|$$

- $g_{\mathcal{H}}(M) \leq 2^M \; \forall \, \mathcal{H}$ and M
- $\mathcal{H}$ **shatters** S if $|\mathcal{H}(S)| = 2^M$
- If $\exists \, S$ s.t. $|S| = M$ and $\mathcal{H}$ shatters S, then $g_{\mathcal{H}}(M) = 2^M$

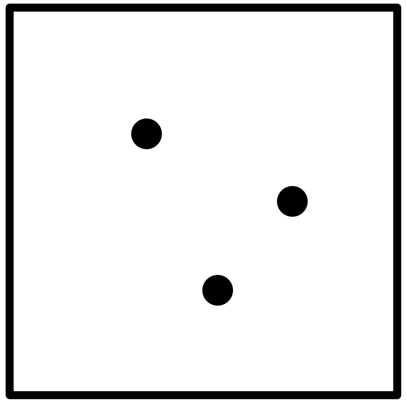
Growth Function: Example

- $\boldsymbol{x}^{(m)} \in \mathbb{R}^2$ and $\mathcal{H}$ = all 2-dimensional linear separators

- $g_{\mathcal{H}}(3) = 8 = 2^3$



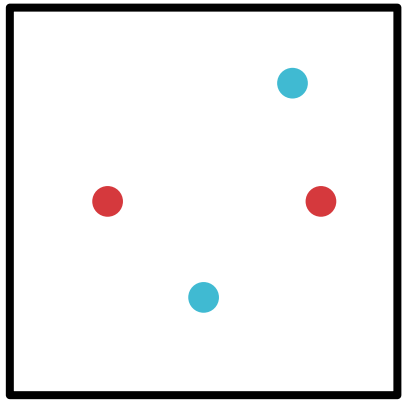
$|\mathcal{H}(S_1)| = 6$



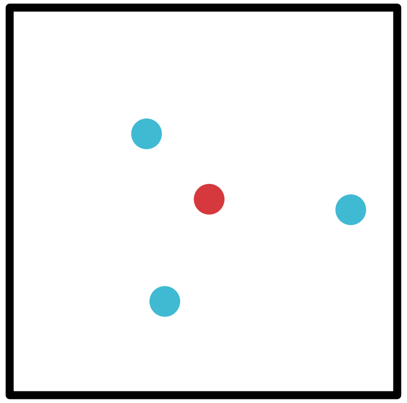
$|\mathcal{H}(S_2)| = 8$

Growth Function: Example

- $\boldsymbol{x}^{(m)} \in \mathbb{R}^2$ and $\mathcal{H}$ = all 2-dimensional linear separators
- $g_{\mathcal{H}}(4) = 14 < 2^4$



$|\mathcal{H}(S_1)| = 14$



$|\mathcal{H}(S_2)| = 14$

Theorem 3: Vapnik-Chervonenkis (VC)-Bound

- Infinite, realizable case: for any hypothesis set $\mathcal{H}$ and distribution p^*, if the number of labelled training data points satisfies

$$M \geq \frac{2}{\epsilon}\left(\log_2\left(2g_{\mathcal{H}}(2M)\right) + \log_2\left(\frac{1}{\delta}\right)\right)$$

then with probability at least $1-\delta$, all $h \in \mathcal{H}$ with $R(h) \geq \epsilon$ have $\hat{R}(h) > 0$

- M appears on both sides of the inequality…

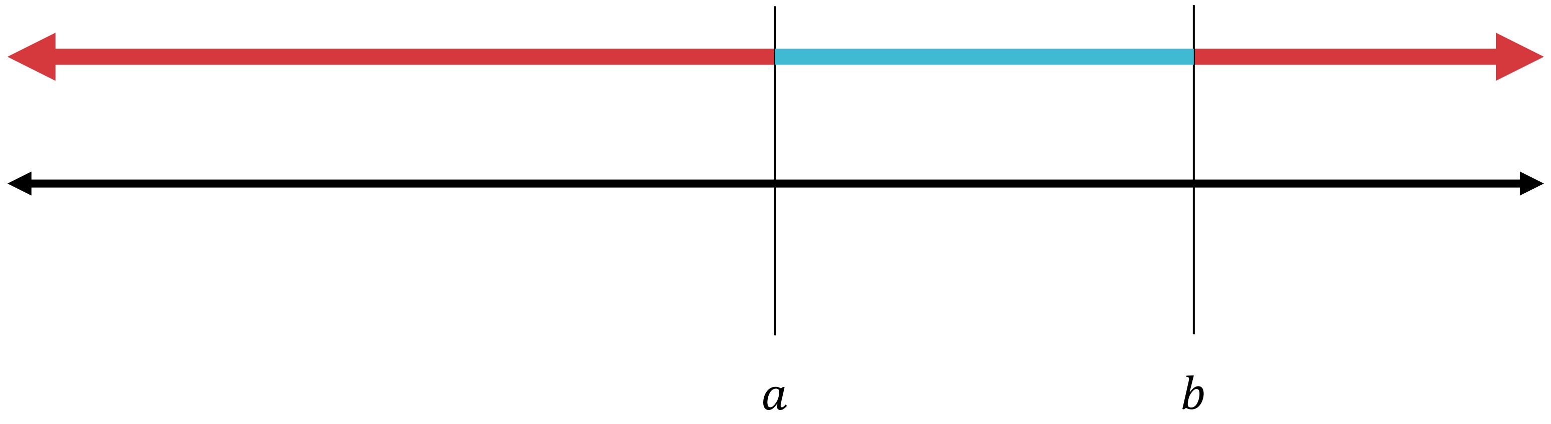
Theorem 3: Vapnik-Chervonenkis (VC)-Dimension

- $d_{VC}(\mathcal{H}) =$ the largest value of M s.t. $g_{\mathcal{H}}(M) = 2^{M}$, i.e., the greatest number of data points that can be shattered by $\mathcal{H}$
  - If $\mathcal{H}$ can shatter arbitrarily large finite sets, then $d_{VC}(\mathcal{H}) = \infty$
  - $g_{\mathcal{H}}(M) = O\left(M^{d_{VC}(\mathcal{H})}\right)$ (Sauer-Shelah lemma)

- To prove that $d_{VC}(\mathcal{H}) = C$, you need to show
  1. $\exists$ some set of C data points that $\mathcal{H}$ can shatter and
  2. $\nexists$ a set of $C + 1$ data points that $\mathcal{H}$ can shatter

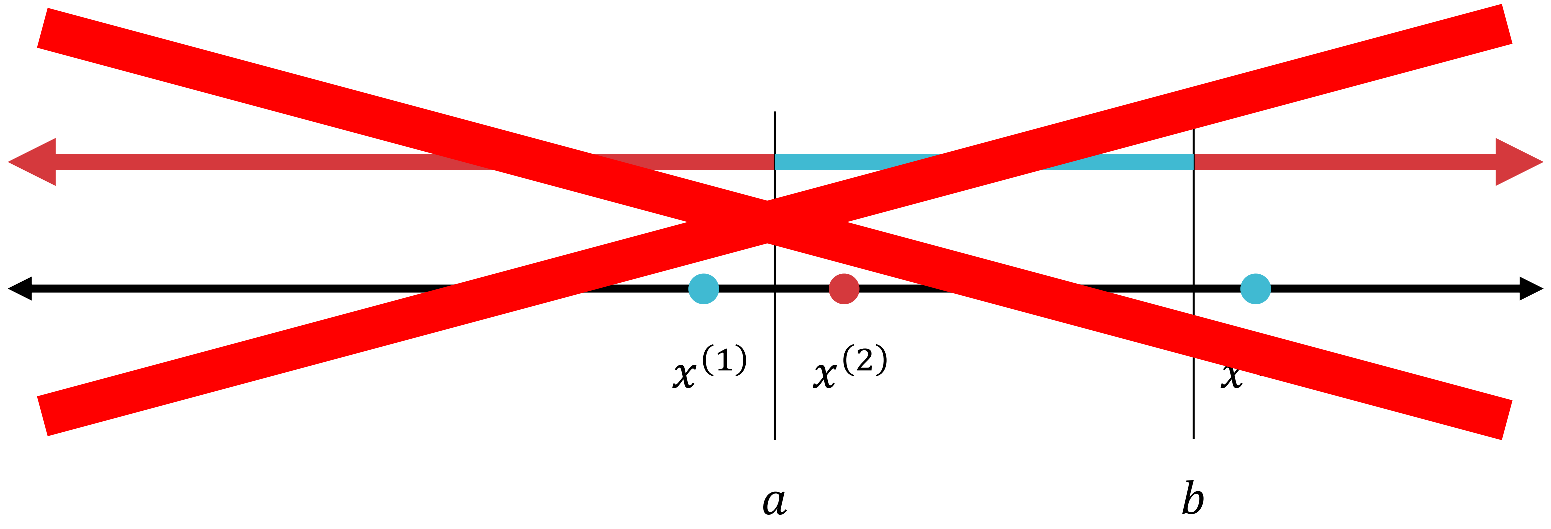
VC-Dimension: Example

- $x^{(m)} \in \mathbb{R}$ and $\mathcal{H}$ = all 1-dimensional positive intervals



VC-Dimension: Example

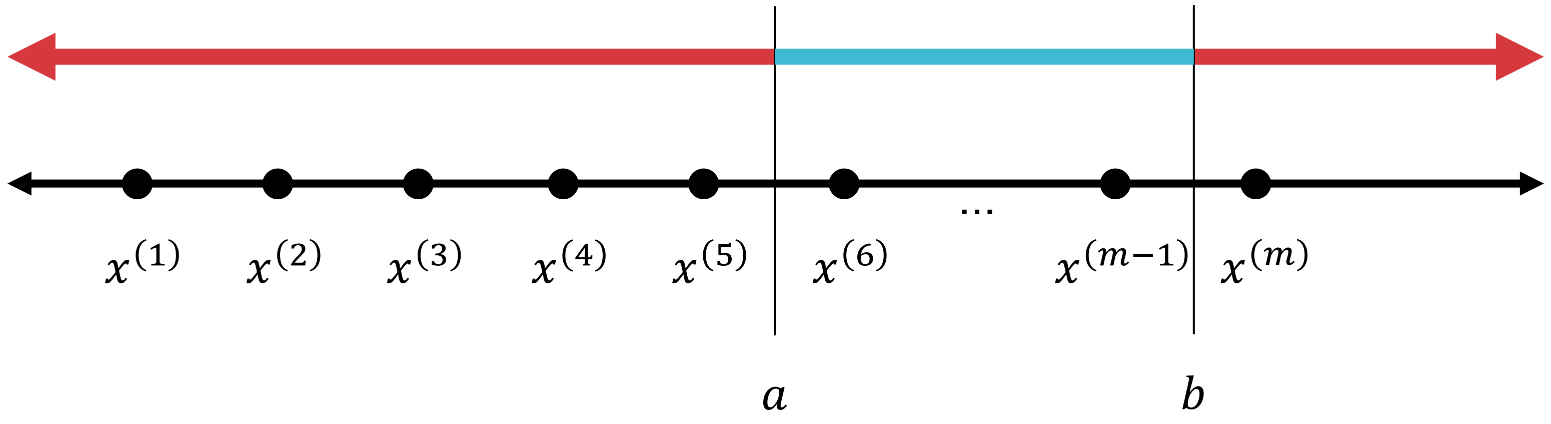
- $x^{(m)} \in \mathbb{R}$ and $\mathcal{H} =$ all 1-dimensional positive intervals



- What are $d_{VC}(\mathcal{H})$ and $g_{\mathcal{H}}(m)$?

VC-Dimension: Example

- $x^{(m)} \in \mathbb{R}$ and $\mathcal{H}$ = all 1-dimensional positive intervals



- $d_{VC}(\mathcal{H}) = 2$ and $g_{\mathcal{H}}(m) = \binom{m+1}{2} + 1 = O(m^2)$

Theorem 3: Vapnik-Chervonenkis (VC)-Bound

- Infinite, realizable case: for any hypothesis set $\mathcal{H}$ and distribution p^*, if the number of labelled training data points satisfies

$$M = O\left(\frac{1}{\epsilon}\left(d_{VC}(\mathcal{H})\log\left(\frac{1}{\epsilon}\right) + \log\left(\frac{1}{\delta}\right)\right)\right)$$

then with probability at least $1 - \delta$, all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have $R(h) \leq \epsilon$

Statistical Learning Theory Corollary

- Infinite, realizable case: for any hypothesis set $\mathcal{H}$ and distribution p^*, given a training data set S s.t. $|S| = M$, all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have

$$R(h) \leq O\left(\frac{1}{M}\left(d_{VC}(\mathcal{H}) \log\left(\frac{M}{d_{VC}(\mathcal{H})}\right) + \log\left(\frac{1}{\delta}\right)\right)\right)$$

with probability at least $1 - \delta$.

Theorem 4: Vapnik-Chervonenkis (VC)-Bound

- Infinite, agnostic case: for any hypothesis set $\mathcal{H}$ and distribution p^*, if the number of labelled training data points satisfies

$$M = O\left(\frac{1}{\epsilon^2}\left(d_{VC}(\mathcal{H}) + \log\left(\frac{1}{\delta}\right)\right)\right)$$

then with probability at least $1 - \delta$, all $h \in \mathcal{H}$ have $\left|R(h) - \hat{R}(h)\right| \leq \epsilon$

Statistical Learning Theory Corollary

- Infinite, agnostic case: for any hypothesis set $\mathcal{H}$ and distribution p^*, given a training data set S s.t. $|S| = M$, all $h \in \mathcal{H}$ have

$$R(h) \leq \hat{R}(h) + O\left(\sqrt{\frac{1}{M}\left(d_{VC}(\mathcal{H}) + \log\left(\frac{1}{\delta}\right)\right)}\right)$$

with probability at least $1 - \delta$.

Approximation Generalization Tradeoff

$$R(h) \leq \underbrace{\hat{R}(h)}_{\text{Decreases as } d_{VC}(\mathcal{H}) \text{ increases}} + O\left(\overbrace{\sqrt{\frac{1}{M}\left(d_{VC}(\mathcal{H}) + \log\left(\frac{1}{\delta}\right)\right)}}^{\text{Increases as } d_{VC}(\mathcal{H}) \text{ increases}}\right)$$