CSE 417T
Introduction to Machine Learning

Lecture 8

Instructor: Chien-Ju (CJ) Ho

Logistics

- HW1: Due Feb 14
  - Reserve time if you have never used Gradescope
  - Make sure the submission is readable (if you scan your handwriting)
  - **Correctly assign pages** to each problem
- HW2: Due Feb 24
- Exams
  - Exam1: March 10 (Thursday)
    - Timed exam at the lecture time
    - More logistical details to come within 1~2 weeks
  - Exam 2: last lecture of the semester

Recap

Linear Models

This is why it's called linear models

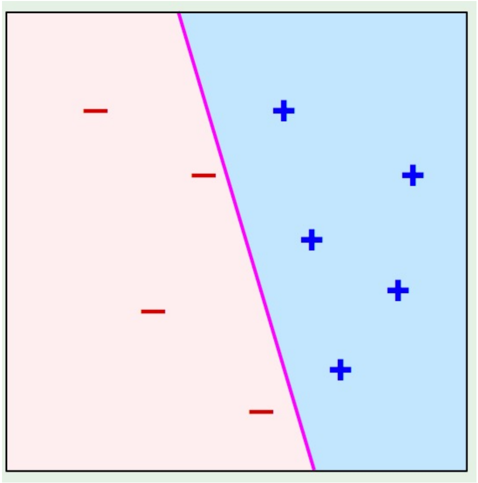
- H contains hypothesis $h(\vec{x})$ as **some function of $\vec{w}^T\vec{x}$**

	Domain	Model	Credit Card Example
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = sign(\vec{w}^T\vec{x})\}$	Approve or not
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$	Credit line
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T\vec{x})\}$	Prob. of default

$$\theta(s) = \frac{e^s}{1 + e^s}$$

- Algorithm:
  - Focus on $g = argmin_{h \in H} E_{in}(h)$

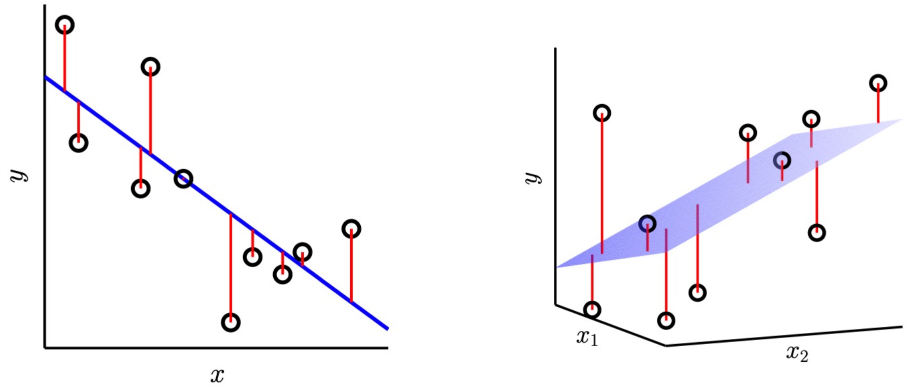
Linear Classification (Perceptron)



- Formulation
  - Hypothesis set $H = \{h(\vec{x}) = sign(\vec{w}^T \vec{x})\}$
  - Error measure: binary error $e(h(\vec{x}), y) = \mathbb{I}[h(\vec{x}) \neq y]$
- Data is linearly separable
  - Run PLA => $E_{in} = 0$ => Low E_{out}
- Data is not linearly separable
  - Minimizing E_{in} is NP hard
  - Pocket algorithm
  - Engineering the features (e.g., handwritten digits)
  - More discussion later in the semester

Linear Regression

- Formulation
  - Hypothesis set $H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$
  - Squared error $e(h(\vec{x}), y) = (h(\vec{x}) - y)^2$



- Given dataset $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
  - $E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N}(\vec{w}^T\vec{x}_n - y_n)^2$

- Goal: find $\vec{w}_{lin} = argmin_{\vec{w}}\, E_{in}(\vec{w})$

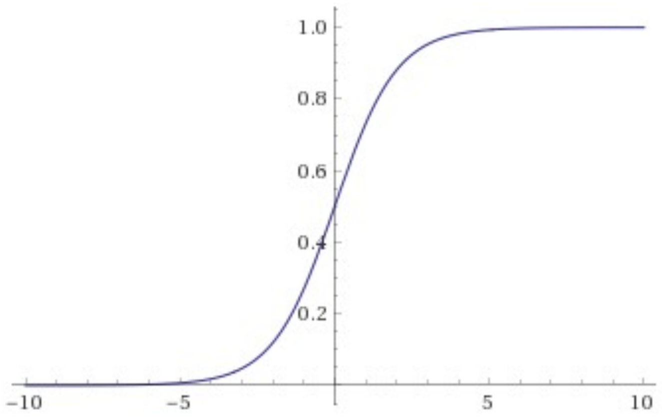
Linear Regression "Algorithm"

- There is a closed-form solution for minimizing E_{in}
  - Closed-form solution for $\nabla_{\vec{w}} E_{in}(\vec{w}) = 0$ (E_{in} is convex)

- One-step algorithm
  - Given $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$
  - Construct $X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,d} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{2,0} & x_{N,1} & \cdots & x_{N,d} \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$
  - Output $\vec{w}_{lin} = (X^T X)^{-1} X^T \vec{y}$ (Assume $X^T X$ is invertible)

Logistic Regression

- Predict a probability
  - Interpreting $h(\vec{x}) \in [0,1]$ as the probability for $y = +1$ given $\vec{x}$
- Hypothesis set $H = \{h(\vec{x}) = \theta(\vec{w}^T \vec{x})\}$
  - $\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$
- Algorithm
  - Find $g = argmin_{h \in H} E_{in}(h)$
- Two key questions
  - How to define $E_{in}(h)$?
  - How to perform the optimization (minimizing E_{in})?



Define $E_{in}(\vec{w})$: Cross-Entropy Error

$$E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$$

- Minimizing cross entropy error is the same as maximizing likelihood

- Likelihood: $\Pr(D|\vec{w})$
  - $\vec{w}^* = argmax_{\vec{w}} \Pr(D|\vec{w})$ (maximizing likelihood)
    $= argmin_{\vec{w}} E_{in}(\vec{w})$ (minimizing cross-entropy error)

Today’s Lecture

The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook.

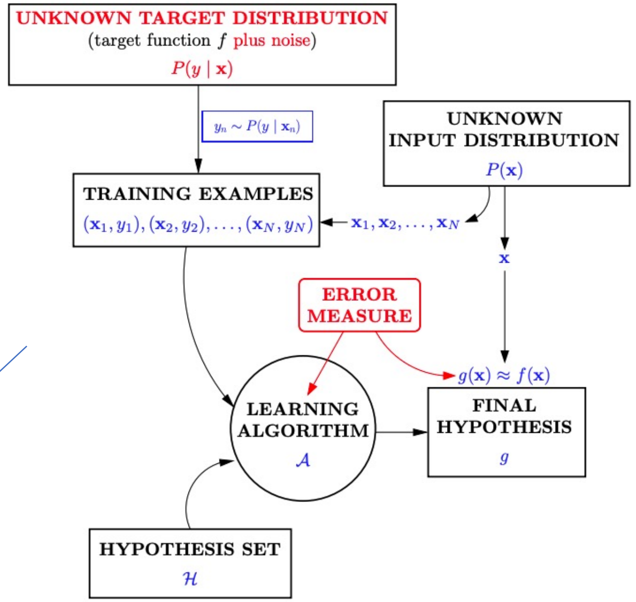
Let me know if you spot errors.

Minimizing Cross Entropy Error

=

Maximizing Likelihood

Write Down the Likelihood



- How are $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ generated?
  - $(\vec{x}_1, \ldots, \vec{x}_N)$ are i.i.d. drawn from a distribution
  - $(y_1, \ldots, y_N)$ are labeled according to target function $f(\vec{x})$

- Likelihood $\Pr(D|h)$
  - The probability of seeing dataset D if D is generated according to h
  - $\Pr(D|h) = \Pr((\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)|h)$

$$= \Pr(\vec{x}_1, \ldots, \vec{x}_N) \Pr\big((y_1, \ldots, y_N)|(\vec{x}_1, \ldots, \vec{x}_N), h\big)$$

$$= \prod_{n=1}^{N} \Pr(\vec{x}_n) \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$$

(Assumption of independent data)

Maximum Likelihood

- Choosing the hypothesis that maximizes the likelihood
  - $g = argmax_{h \in H} \Pr(D|h)$
    $= argmax_{h \in H} \prod_{n=1}^{N} \Pr(\vec{x}_n) \ \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$
    $= argmax_{h \in H} \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$

$\prod_{n=1}^{N} \Pr(\vec{x}_n)$ doesn't depend on h

- We interpret $h(\vec{x})$ as the probability of $y = +1$
  - $\Pr(y|\vec{x}, h) = \begin{cases} h(\vec{x}) = \theta(\vec{w}^T \vec{x}) & \text{for } y = +1 \\ 1 - h(\vec{x}) = 1 - \theta(\vec{w}^T \vec{x}) & \text{for } y = -1 \end{cases}$

- Since $1 - \theta(s) = \theta(-s)$
  - $\Pr(y|\vec{x}, h) = \theta(y\,\vec{w}^T \vec{x})$

Maximum Likelihood

- Choosing the hypothesis that maximizes the likelihood
  - $g = argmax_{h \in H} \Pr(D|h)$
    $= argmax_{h \in H} \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$

  - $\vec{w}^* = argmax_{\vec{w}} \prod_{n=1}^{N} \theta(y_n \vec{w}^T \vec{x}_n)$
    $= argmax_{\vec{w}} \ln(\prod_{n=1}^{N} \theta(y_n \vec{w}^T \vec{x}_n))$
    $= argmax_{\vec{w}} \sum_{n=1}^{N} \ln(\theta(y_n \vec{w}^T \vec{x}_n))$
    $= argmin_{\vec{w}} - \sum_{n=1}^{N} \ln(\theta(y_n \vec{w}^T \vec{x}_n))$
    $= argmin_{\vec{w}} \sum_{n=1}^{N} \ln \frac{1}{\theta(y_n \vec{w}^T \vec{x}_n)}$
    $= argmin_{\vec{w}} \sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$
    $= argmin_{\vec{w}} \boxed{\frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})}$

$$\boxed{\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}}$$

$$E_{in}(\vec{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$$

Cross Entropy Error

$$E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$$

- Minimizing $E_{in}(\vec{w})$ is the same as maximizing likelihood
- Next question: How to solve $\vec{w}^* = argmin_{\vec{w}} E_{in}(\vec{w})$
  - Answer: Solve for $\nabla_{\vec{w}} E_{in}(\vec{w}) = 0$
  - No single-step solution like we have in linear regression

Using Logistic Regression for Classification

- Let $\vec{w}^*$ or g be the learned logistic regression model, how can we make classification predictions using $\vec{w}^*$?

- Set a cutoff probability C% (e.g., 50%).
  - Classify +1 if $g(\vec{x}) = \theta({\vec{w}^*}^T \vec{x}) > C\%$
  - Classify -1 if $g(\vec{x}) = \theta({\vec{w}^*}^T \vec{x}) < C\%$

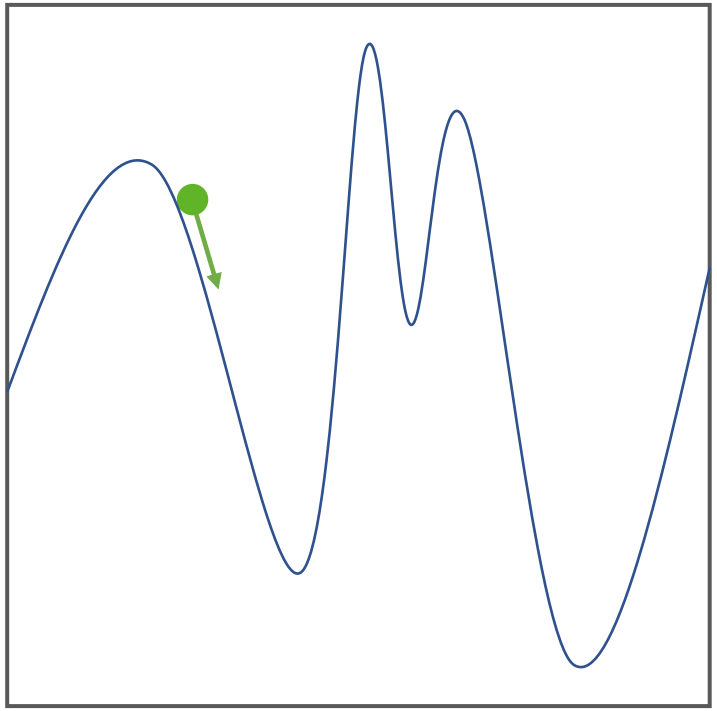
- When C is 50 (a common choice)
  - $\theta({\vec{w}^*}^T \vec{x}) > 50\%$ => ${\vec{w}^*}^T \vec{x} > 0$
  - Equivalent to using $\vec{w}^*$ as the linear classification hypothesis, i.e., $g(\vec{x}) = sign({\vec{w}^*}^T \vec{x})$

Gradient Descent

A general optimization technique

Gradient Descent

- A technique for optimizing functions that gradients exist everywhere.



- An iterative method that converges to local optimum.
- Converge to global optimum if the function is convex (since there is only one local optimum).

Gradient Descent: Minimizing $E_{in}(\vec{w})$

- An iterative method of the form:

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$$

- $\vec{v}_t$: a unit vector, determining the direction of the update
- η_t: a scalar, determining how much to update

- How to choose $\vec{v}_t$ and η_t?

Choosing $\vec{v}_t$ in $\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

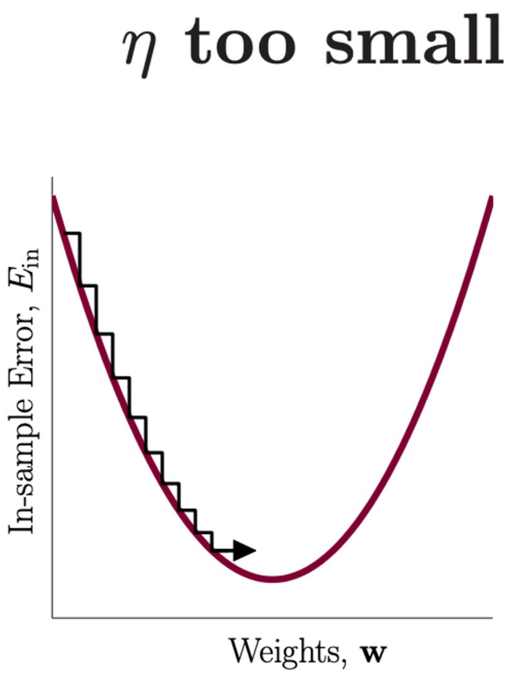
- Intuition: Choose $\vec{v}_t$ that moves towards the "steepest" direction
  - Approaching the minimum faster

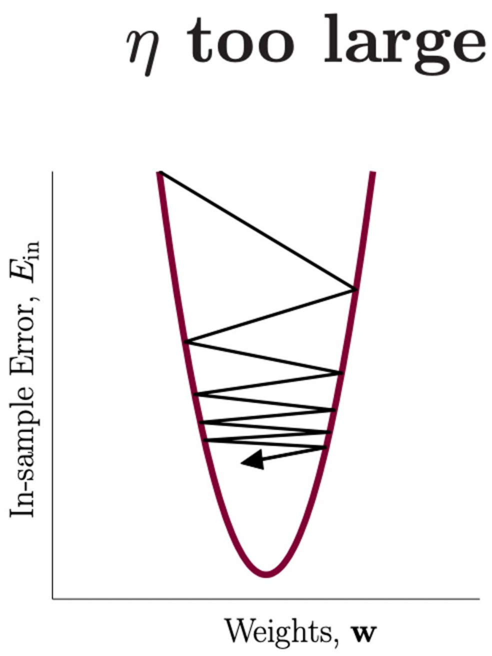
- Taylor's approximation:
  - $E_{in}(\vec{w}(t) + \eta_t \vec{v}_t) = E_{in}(\vec{w}(t)) + \eta_t \nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t + O(\eta_t^2)$
  - $E_{in}(\vec{w}(t+1)) - E_{in}(\vec{w}(t)) \approx \eta_t \nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t$

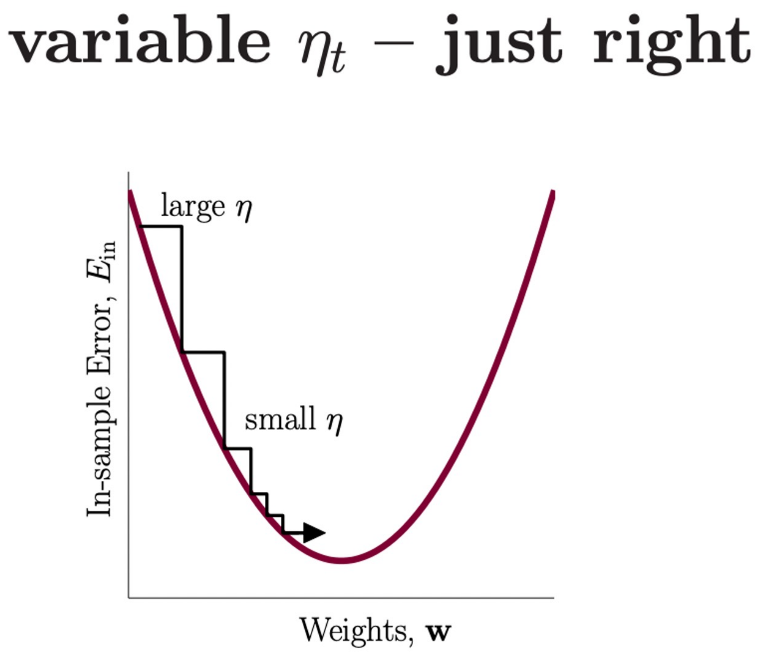
η_t is usually small, so ignore this term

- Choose unit vector $\vec{v}_t$ that minimizes $\nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t$
  - $\vec{v}_t$ should be in the opposite direction of $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$
  - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$

Choosing η_t in $\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

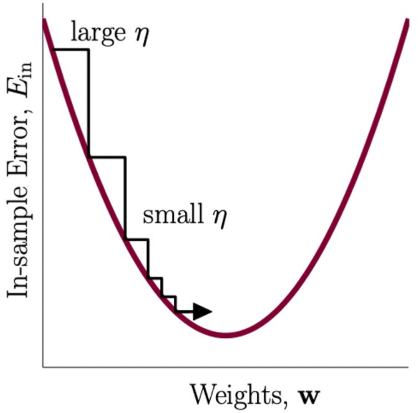






Choosing η_t in $\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

- Intuition (for convex E_{in})
  - When E_{in} is closer to the minimum,
    - $\nabla_{\vec{w}} E_{in}\left(\vec{w}(t)\right)$ is smaller
    - We should set η_t smaller



- Therefore, set $\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$

Putting Them Together

$$\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}\big(\vec{w}(t)\big)}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$$

$$\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$$

- Iterative update rule: $\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$
  - $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta\, \nabla_{\vec{w}} E_{in}\big(\vec{w}(t)\big)$
- Gradient calculations
  - $E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$
  - $\nabla_{\vec{w}} E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \frac{-y_n \vec{x}\, e^{-y_n \vec{w}^T \vec{x}_n}}{1+e^{-y_n \vec{w}^T \vec{x}_n}} = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}^T \vec{x}_n}}$

Gradient Descent for Logistic Regression

- Initialize $\vec{w}(0)$
- For $t = 0, \ldots$
    - Compute $\nabla_{\vec{w}} E_{in}(\vec{w}(t)) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}(t)^T \vec{x}_n}}$
    - $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} E_{in}\left(\vec{w}(t)\right)$
    - Terminate if the stop conditions are met
- Return the final weights

η: learning rate.
A parameter the learner can choose.

Gradient Descent for Logistic Regression

- Initialization
  - Random initialization is a good idea and a common approach
  - (we specify the initialization in HW2 mostly for grading purposes)

- Stop conditions (a mix of the following criteria)
  - When the number of iteration exceeds the pre-set threshold
  - When the improvement on E_{in} (e.g., check $\nabla_{\vec{w}} E_{in}$) is too small
  - When E_{in} is small enough

Computation of Gradient Descent

- Gradient Descent for Logistic Regression
  - Initialize $\vec{w}(0)$
  - For $t = 0, \ldots$
    - Compute $\nabla_{\vec{w}} E_{in}(\vec{w}(t)) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}(t)^T \vec{x}_n}}$
    - $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} E_{in}\left(\vec{w}(t)\right)$
    - Terminate if the stop conditions are met
  - Return the final weights

- Which step requires the most computation?
  - Calculate $\nabla_{\vec{w}} E_{in}(\vec{w}) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}^T \vec{x}_n}}$
  - The time complexity is $O(N)$
    - N is large for big datasets

Stochastic Gradient Descent

Deal with Heavy Computation of $\nabla_{\vec{w}} E_{in}(\vec{w})$

- Speed up the implementation of $\nabla_{\vec{w}} E_{in}(\vec{w}) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}^T \vec{x}_n}}$
  - Vectorization can make your HW2 running time in several order of magnitudes faster
  - Example:
    - Given $[x_1, \dots, x_N]$, want to calculate $[e^{x_1}, \dots, e^{x_N}]$
    - Using for loop:
      - Loop from n=1 to N, calculate e^{x_n}
    - Vectorized method:
      - Using numpy library: np.exp($[x_1, \dots, x_N]$)
  - Why? Matrix operations are optimized in a low level using numpy operations (or other scientific computing libraries).
    - Try to replace loops with numpy matrix operations in your HW2

Deal with Heavy Computation of $\nabla_{\vec{w}} E_{in}(\vec{w})$

- Speed up the implementation of $\nabla_{\vec{w}} E_{in}(\vec{w}) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}^T \vec{x}_n}}$
  - Vectorization can make your HW2 running time in several order of magnitudes faster

- Solve $\nabla_{\vec{w}} E_{in}(\vec{w})$ "in expectation"
  - Define $e_n(\vec{w}) = \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$, the point-wise error caused by $(\vec{x}_n, y_n)$
  - Observe that
    - $E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} e_n(\vec{w})$
    - $\nabla_{\vec{w}} E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \nabla_{\vec{w}} e_n(\vec{w})$
    - Draw a point $\vec{x}_n$ from $\{\vec{x}_1, \dots, \vec{x}_N\}$ uniformly at random
      - $E_{\vec{x}_n}[\nabla_{\vec{w}} e_n(\vec{w})] = \nabla_{\vec{w}} E_{in}(\vec{w})$

Stochastic Gradient Descent (SGD)

- Algorithm
  - Initialize $\vec{w}(0)$
  - For $t = 0, \dots$
    - Randomly choose n from $\{1, \dots, N\}$
    - $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} e_n(\vec{w}(t))$
    - Terminate if the stop conditions are met
  - Return the final weights

- $\mathbb{E}[\nabla_{\vec{w}} e_n(\vec{w})] = \nabla_{\vec{w}} E_{in}(\vec{w})$
  - SGD is doing the same thing as GD **in expectation**
    - More efficient (scale to large dataset), suitable for online data, helps escaping local min, etc.
    - Noisier, harder to define stop criteria

Mini-Batch Gradient Descent

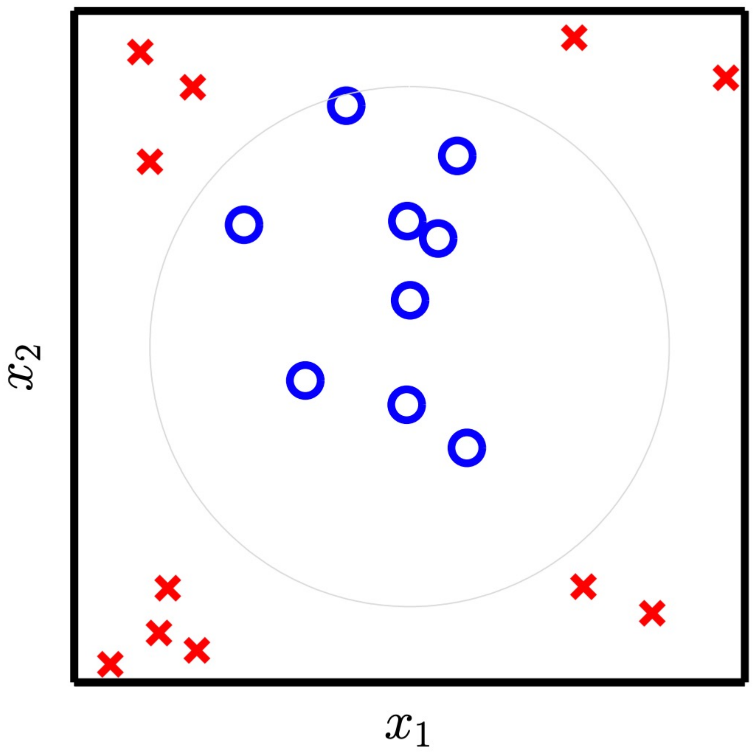
- GD: Computationally heavy, stable updates
- SGD: Computationally light, noisy updates

- Middle ground: Mini-Batch Gradient Descent
  - In each iteration, randomly choose k points $\{n(1), \ldots, n(k)\}$
  - Update rule
    - $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta \frac{1}{k} \sum_{i=1}^{k} \nabla_{\vec{w}} e_{n(i)}(\vec{w}(t))$

- Side note about HW2
  - Please report your results on GD (non-stochastic version)
    - You should feel free to play around with SGD or mini-batch on your own

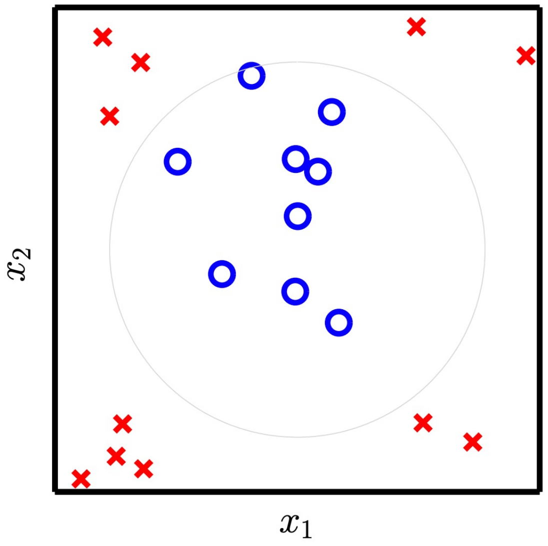
Non-Linear Transformation

Limitations of Linear Models



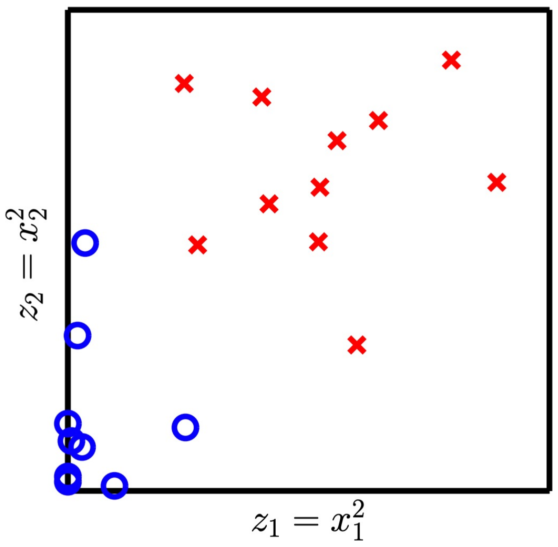
Using Non-Linear Transformations

- Find a feature transform Φ that maps data from $\vec{x}$ space to $\vec{z}$ space



$\vec{x} = (1, x_1, x_2)$

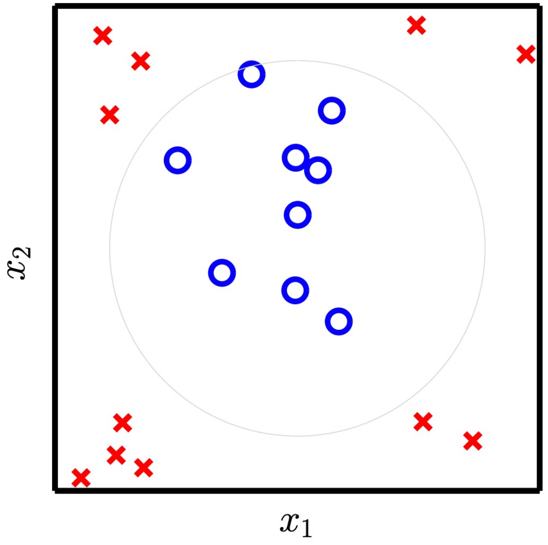
$\vec{z} = \Phi(\vec{x})$



$\vec{z} = (1, x_1^2, x_2^2)$

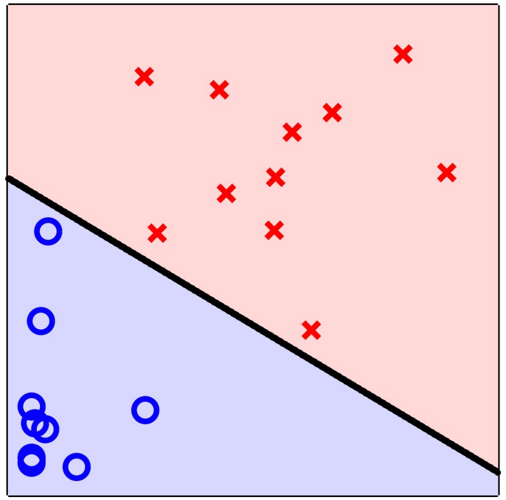
Using Non-Linear Transformations

- Learn a linear classifier in $\vec{z}$ space: $g^{(z)}(\vec{z}) = sign\left({\vec{w}^{(z)}}^T \vec{z}\right)$



$\vec{x} = (1, x_1, x_2)$

$\vec{z} = \Phi(\vec{x})$

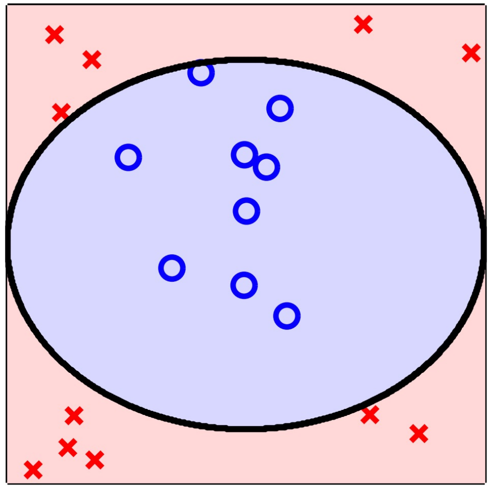


$\vec{z} = (1, x_1^2, x_2^2)$

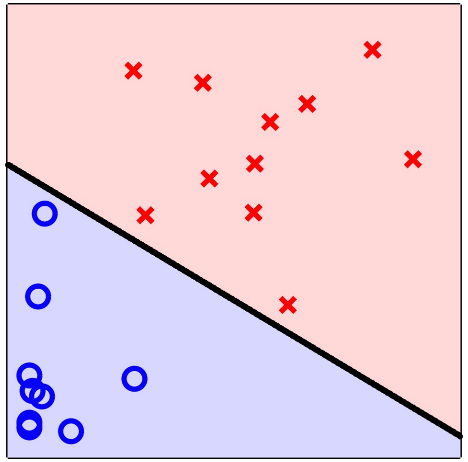
$g^{(z)}(\vec{z}) = sign(-0.6 + z_1 + z_2)$

Using Non-Linear Transformations

- Transform the learned hypothesis back to $\vec{x}$ space
  - $g(\vec{x}) = g^{(z)}(\Phi(\vec{x})) = sign\left(\vec{w}^{(z)^T}\Phi(\vec{x})\right)$



$\vec{x} = (1, x_1, x_2)$

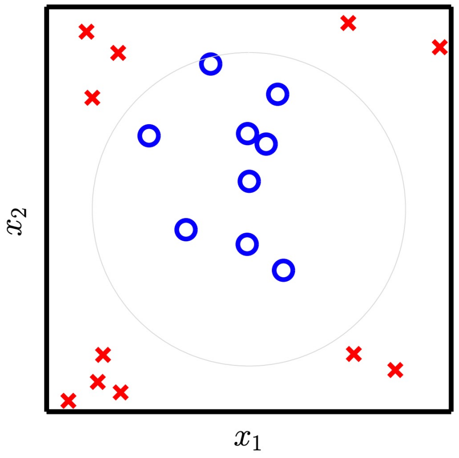


$\vec{z} = (1, x_1^2, x_2^2)$

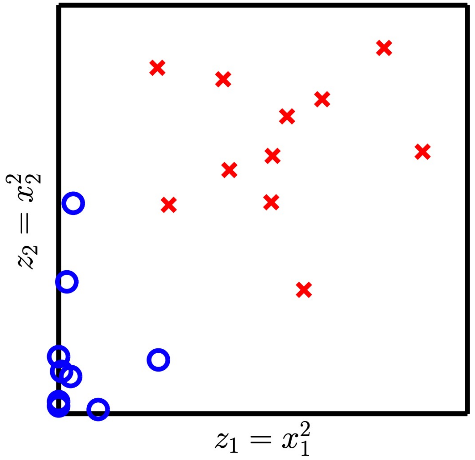
$g(\vec{x}) = sign(-0.6 + x_1^2 + x_2^2)$

$g^{(z)}(\vec{z}) = sign(-0.6 + z_1 + z_2)$

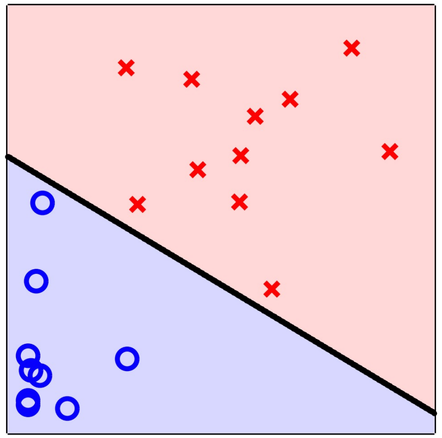
Nonlinear Transformation



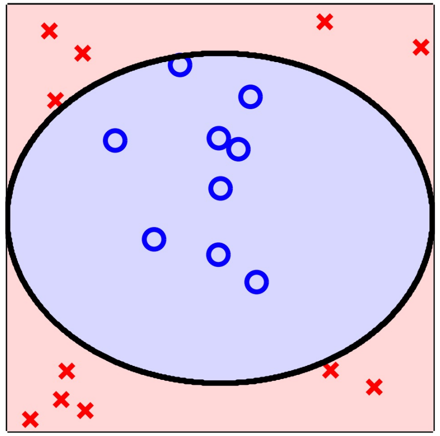
$$\vec{z} = \Phi(\vec{x})$$



$$g^{(z)}(\vec{z}) = sign({\vec{w}^{(z)}}^T \vec{z})$$

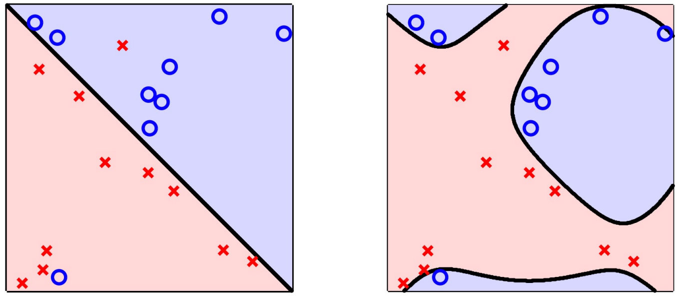


$$g(\vec{x}) = g^{(z)}(\Phi(\vec{x})) = sign({\vec{w}^{(z)}}^T \Phi(\vec{x}))$$

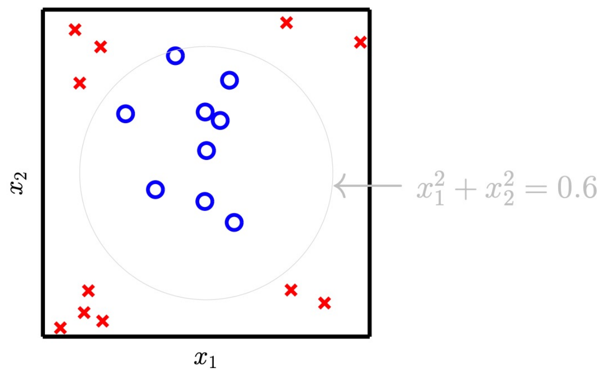


Generalization of Nonlinear Transformation

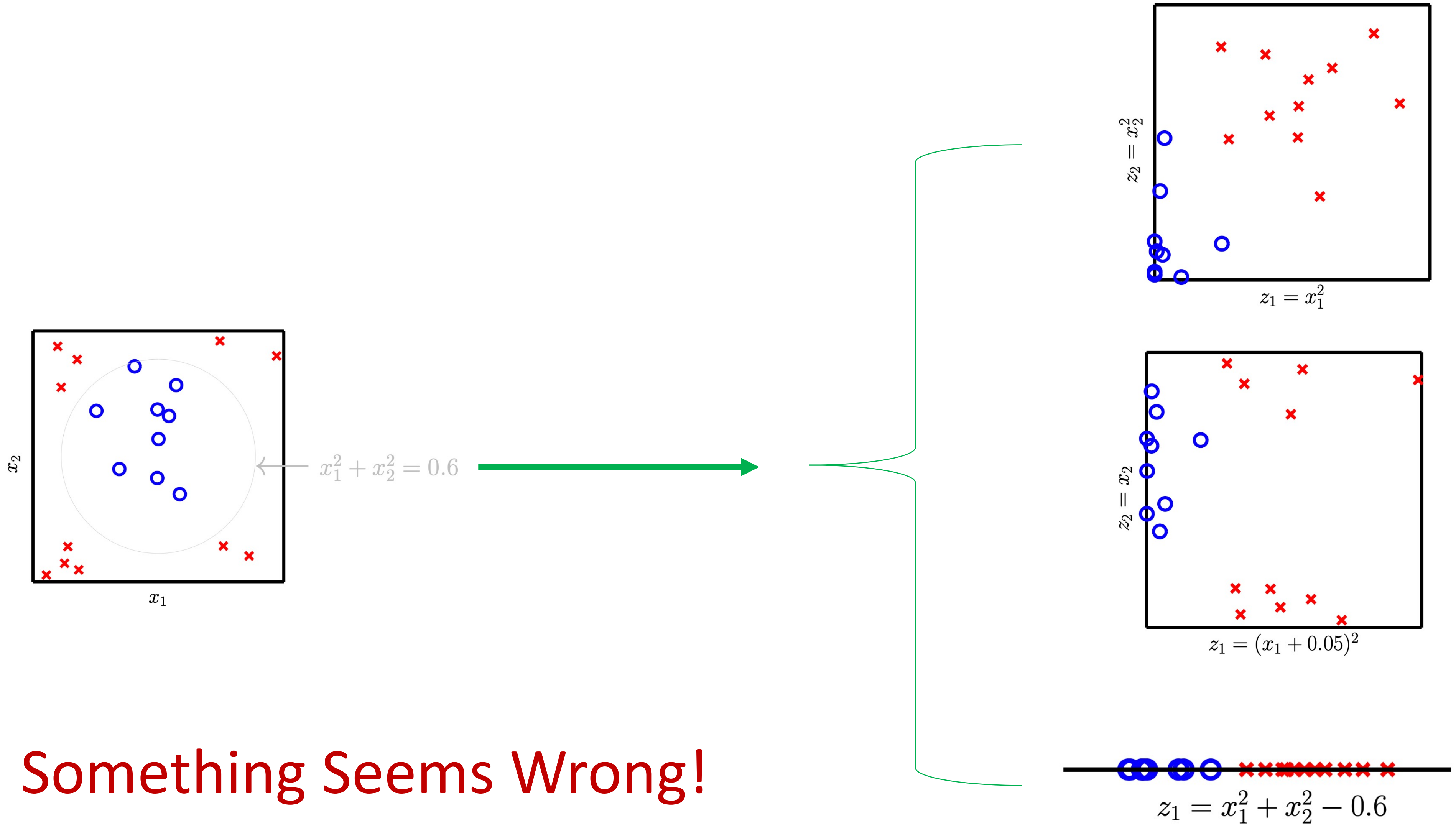
- Fact (We'll prove this later)
  - The VC Dimension of d-dim perceptron is $d + 1$
- VC dimension of perceptron on input space $\vec{x} = (x_0, .., x_d)$
  - d+1
- VC dimension of perceptron on input space $\vec{z} = (z_0, .., z_{d'})$
  - $\leq d' + 1$ (usually treated as $\approx d' + 1$)
- Careful: Non-linear transform might lead to "nonsense" behavior



How to Choose Feature Transform Φ



How to Choose Feature Transform Φ



Something Seems Wrong!

Must choose Φ BEFORE looking at the data

Otherwise, you are doing "data snooping"

The hypothesis set H is as large as anything your brain can think of

Choose Φ Before Seeing Data

- Rely on domain knowledge (feature engineering)
  - Handwriting digit recognition example

- Use common sets of feature transformation
  - Polynomial transformation
  - 2nd order Polynomial transformation
    - $\vec{x} = (1, x_1, x_2)$
    - $\Phi_2(\vec{x}) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$
    - Pros: more powerful (contains circle, ellipse, hyperbola, etc)
    - Cons: 2-d => 5-d
      - More computation/storage
      - Worse generalization error

The VC dimension of d-dim perceptron is d+1

Q-th Order Polynomial Transform

- $\vec{x} = (1, x_1, \dots, x_d)$
- From 1-st order to Q-th order polynomial transform:
  - $\Phi_1(\vec{x}) = \vec{x}$
  - $\Phi_2(\vec{x}) = (\Phi_1(\vec{x}), x_1^2, x_1 x_2, x_1 x_3, \dots, x_d^2)$
  - …
  - $\Phi_Q(\vec{x}) = (\Phi_{Q-1}(\vec{x}), x_1^Q, x_1^{Q-1} x_2, \dots, x_d^Q)$
- Number of elements in $\Phi_Q(\vec{x})$

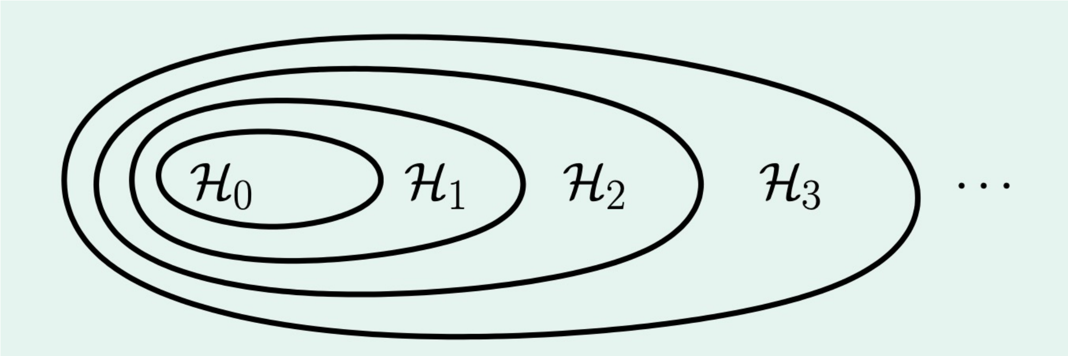
Q-th Order Polynomial Transform

- $\vec{x} = (1, x_1, \dots, x_d)$
- From 1-st order to Q-th order polynomial transform:
  - $\Phi_1(\vec{x}) = \vec{x}$
  - $\Phi_2(\vec{x}) = (\Phi_1(\vec{x}), x_1^2, x_1x_2, x_1x_3, \dots, x_d^2)$
  - …
  - $\Phi_Q(\vec{x}) = (\Phi_{Q-1}(\vec{x}), x_1^Q, x_1^{Q-1}x_2, \dots, x_d^Q)$

- Number of elements in $\Phi_Q(\vec{x})$
  - $\binom{Q+d}{Q}$

Structural Hypothesis Sets

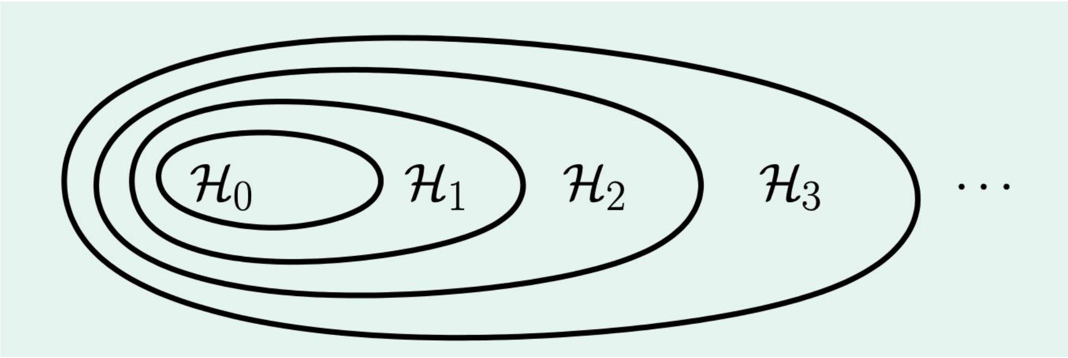
- Let H_Q be the linear model for the $\Phi_Q(\vec{x})$ space



- Let $g_Q = argmin_{h \in H_Q} E_{in}(h)$
  - H_0 H_1 H_2 …
  - $d_{vc}(H_0)$ $d_{vc}(H_1)$ $d_{vc}(H_2)$ …
  - $E_{in}(g_0)$ $E_{in}(g_1)$ $E_{in}(g_2)$ …

Structural Hypothesis Sets

- Let H_Q be the linear model for the $\Phi_Q(\vec{x})$ space



- Let $g_Q = argmin_{h \in H_Q} E_{in}(h)$
  - $H_0 \subset H_1 \subset H_2 \ldots$
  - $d_{vc}(H_0) \leq d_{vc}(H_1) \leq d_{vc}(H_2) \ldots$
  - $E_{in}(g_0) \geq E_{in}(g_1) \geq E_{in}(g_2) \ldots$