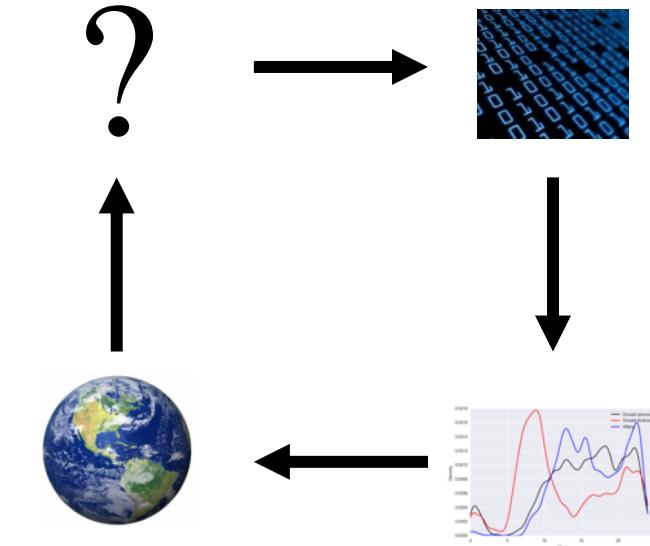


Fitting Linear Models, Regularization (revisited) & Cross Validation

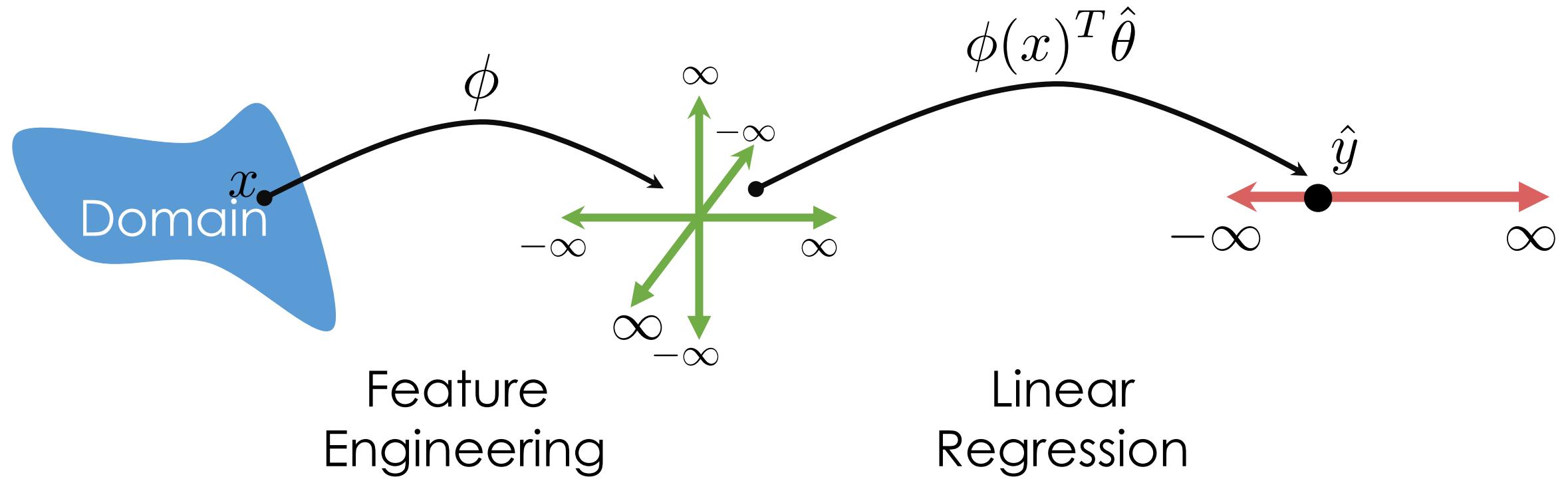
Slides by:

Joseph E. Gonzalez jegonzal@cs.berkeley.edu



Previously

Feature Engineering and Linear Regression

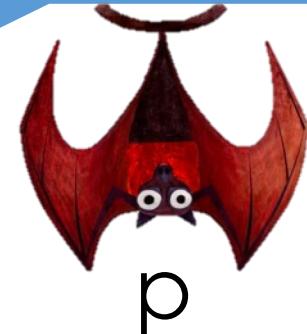


Recap: Feature Engineering

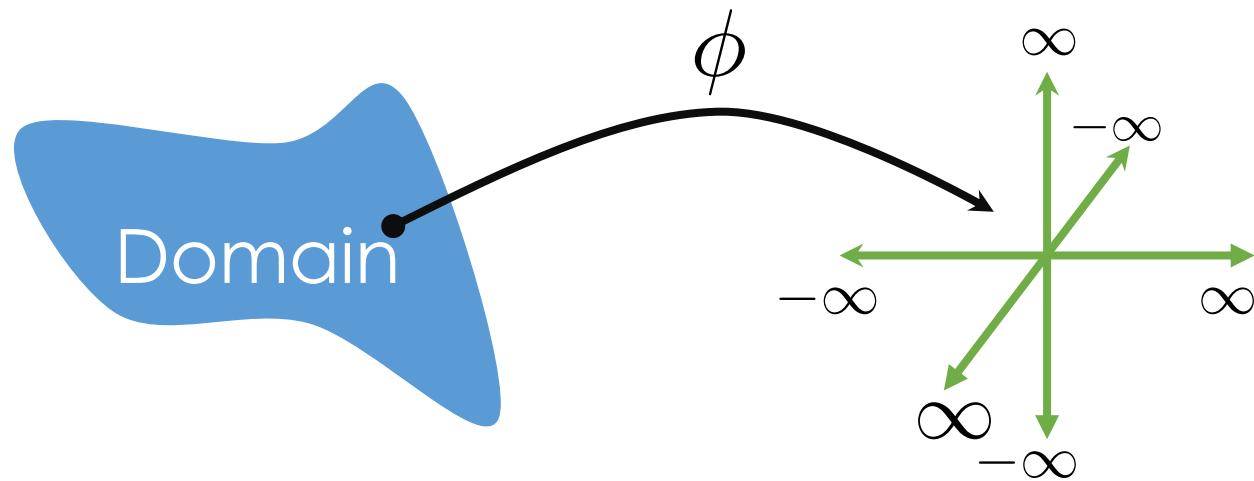
- Linear models with feature functions:

$$f_{\theta}(x) = \sum_{j=1}^d \theta_j \phi_j(x)$$

Notation: Computer scientist / ML researchers tend to use d (dimensions) and statisticians will use p (parameters).



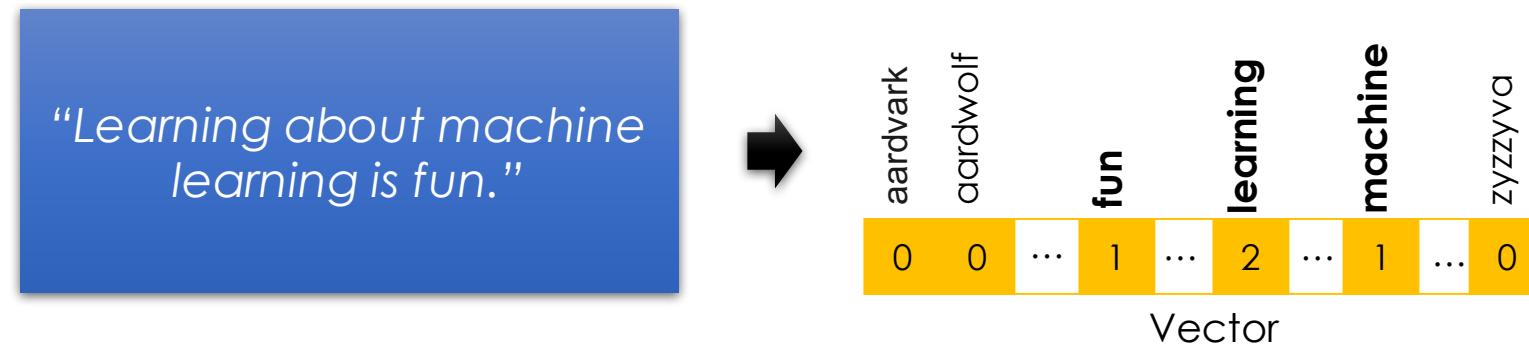
- Feature Functions: $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$



➤ One-hot encoding: Categorical Data

state	AL	...	CA	...	NY	...	WA	...	WY
NY	0	...	0	...	1	...	0	...	0
WA	0	...	0	...	0	...	1	...	0
CA	0	...	1	...	0	...	0	...	0

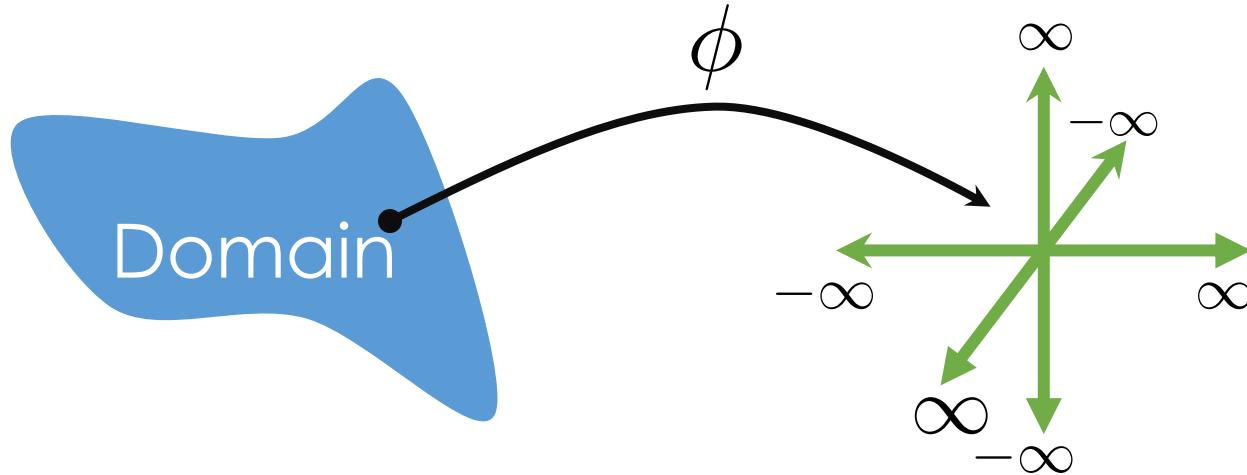
➤ Bag-of-words & N-gram: Text Data



➤ Custom Features: Domain Knowledge

$$\phi(\text{lat}, \text{lon}, \text{amount}) = \frac{\text{amount}}{\text{Stores}[\text{ZipCode}[\text{lat}, \text{lon}]]}$$

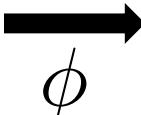
The Feature Matrix Φ



X DataFrame

uid	age	state	hasBought	review
0	32	NY	True	"Meh."
42	50	WA	True	"Worked out of the box ..."
57	16	CA	NULL	"Hella tots lit..."

$$\Phi \in \mathbb{R}^{n \times d}$$



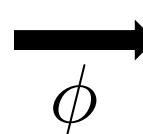
AK	...	NY	...	WY	age	hasBought	hasBought missing
0	...	1	...	0	32	1	0
0	...	0	...	0	50	1	0
0	...	0	...	0	16	0	1

Entirely **Quantitative** Values

X DataFrame

uid	age	state	hasBought	review
0	32	NY	True	"Meh."
42	50	WA	True	"Worked out of the box ..."
57	16	CA	NULL	"Hella tots lit..."

$\Phi \in \mathbb{R}^{n \times d}$



AK	...	NY	...	WY	age	hasBought	hasBought missing
0	...	1	...	0	32	1	0
0	...	0	...	0	50	1	0
0	...	0	...	0	16	0	1

Entirely **Quantitative** Values

Another quick note on confusing notation.

In many textbooks and even in the class notes and discussion you will see:

$$X \in \mathbb{R}^{n \times d} \quad \text{and} \quad \hat{\theta} = (X^T X)^{-1} X^T Y$$

In this case we are assuming X is the transformed data Φ . This can be easier to read but hides the feature transformation process.

Capital Letter: Matrix or Random Variable?

- Both tend to be capitalized
- Unfortunately, there is no common convention ... you will have to use context.

The Feature Matrix Φ

AK	...	NY	...	WY	age	hasBought	hasBought missing
0	...	1	...	0	32	1	0
0	...	0	...	0	50	1	0
0	...	0	...	0	16	0	1

Entirely **Quantitative** Values

$$\Phi \in \mathbb{R}^{n \times d} = \phi(X) = \begin{matrix} \text{DataFrame} \\ \uparrow \end{matrix} \quad \left[\begin{array}{c} \overbrace{\quad \quad \quad}^n \phi(x^{(1)}) \\ \overbrace{\quad \quad \quad}^n \phi(x^{(2)}) \\ \dots \\ \overbrace{\quad \quad \quad}^d \phi(x^{(n)}) \end{array} \right]$$

Rows of the Φ matrix correspond to records.

Columns of the Φ matrix correspond to features.

Confusing notation!

The Feature Matrix Φ

AK	...	NY	...	WY	age	hasBought	hasBought missing
0	...	1	...	0	32	1	0
0	...	0	...	0	50	1	0
0	...	0	...	0	16	0	1

Entirely **Quantitative** Values

$$\Phi \in \mathbb{R}^{n \times d} = \phi(X) = \begin{matrix} n \\ \left[\begin{array}{c} \phi(X_{1,\bullet}) \\ \phi(X_{2,\bullet}) \\ \dots \\ \phi(X_{n,\bullet}) \end{array} \right] \end{matrix}$$

DataFrame

Rows of the Φ matrix correspond to records.

Columns of the Φ matrix correspond to features.

Notation Guide

$A_{i,\bullet}$: row i of matrix A.

$A_{\bullet,j}$: column j of matrix A.

Making Predictions

$$\Phi \in \mathbb{R}^{n \times d} = \phi(X) = \begin{array}{c} \text{DataFrame} \\ \left[\begin{array}{c} \phi(X_1, \bullet) \\ \phi(X_2, \bullet) \\ \vdots \\ \phi(X_n, \bullet) \end{array} \right] \end{array}$$

The diagram shows a matrix Φ with n rows and d columns. The rows are labeled $\phi(X_1, \bullet)$, $\phi(X_2, \bullet)$, \dots , and $\phi(X_n, \bullet)$. A red bracket on the left indicates the number of rows n , and a red bracket at the bottom indicates the number of columns d .

Rows of the Φ matrix correspond to records.

Columns of the Φ matrix correspond to features.

Prediction

$$\hat{Y} = f_{\hat{\theta}}(X) = \Phi \hat{\theta} = \left[\begin{array}{c} \phi(X_1, \bullet) \\ \phi(X_2, \bullet) \\ \vdots \\ \phi(X_n, \bullet) \end{array} \right] \left| \begin{array}{c} \hat{\theta} \end{array} \right| = \left[\begin{array}{c} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(n)} \end{array} \right]$$

The diagram shows the prediction process. The input matrix Φ has rows $\phi(X_1, \bullet)$, $\phi(X_2, \bullet)$, \dots , and $\phi(X_n, \bullet)$. The column vector $\hat{\theta}$ is multiplied by the matrix Φ to produce the output vector \hat{Y} , which contains elements $\hat{y}^{(1)}$, $\hat{y}^{(2)}$, \dots , and $\hat{y}^{(n)}$. The row $\phi(X_2, \bullet)$ is highlighted in orange, and the corresponding element $\hat{y}^{(2)}$ in the output vector is also highlighted in orange.

Normal Equations

- Solution to the least squares model:

$$\hat{\theta} = \arg \min \frac{1}{n} \sum_{i=1}^n \left(y_i - \sum_{j=1}^d \theta_j \phi_j(x_i) \right)^2$$

- Given by the normal equation:

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T Y$$

- **You should know this!**
- You do not need to know the calculus based derivation.
- **You should know the geometric derivation ...**

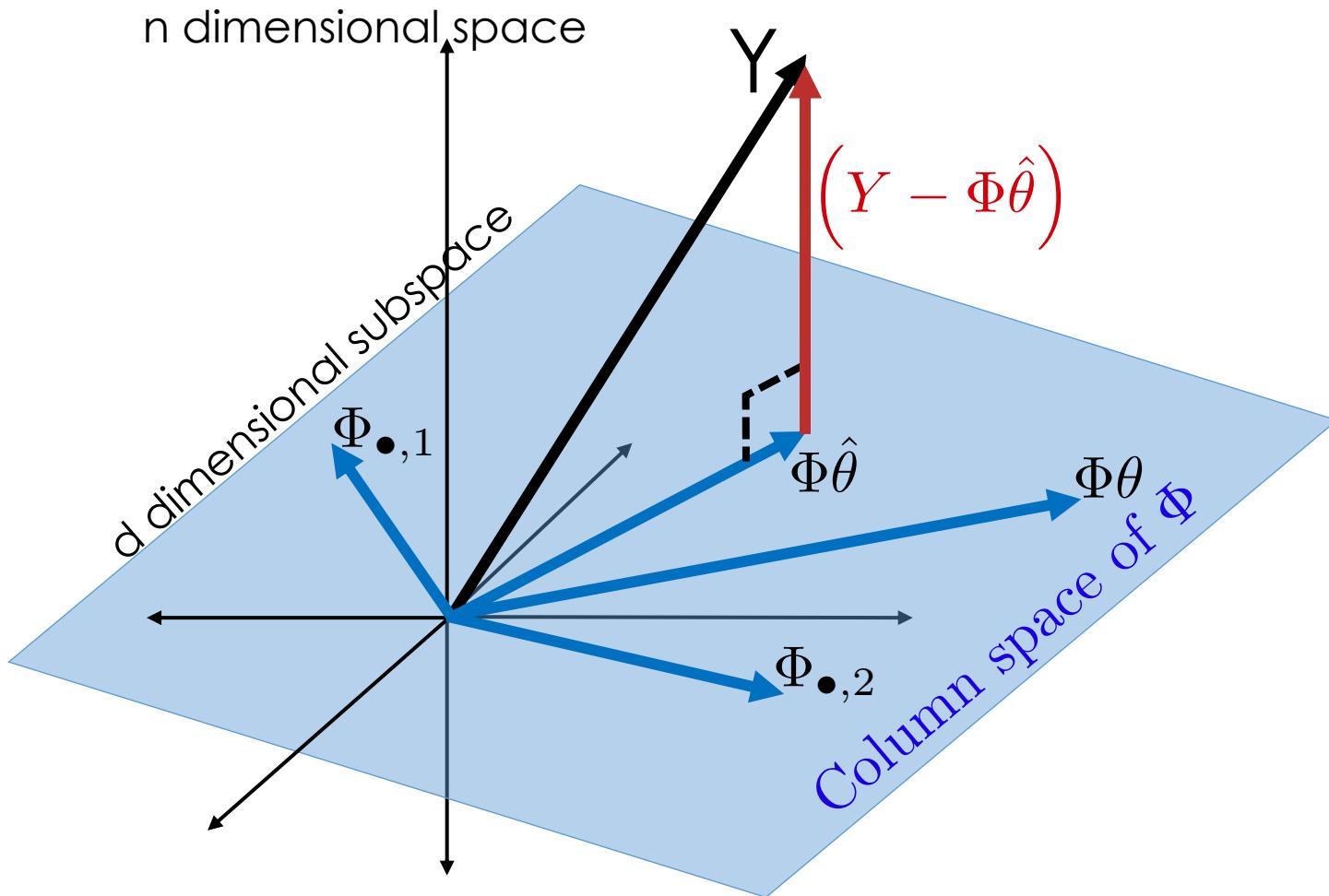
Geometric Derivation: Not Bonus Material

We decided you that this is too exciting to not know.

- Examine the column spaces:

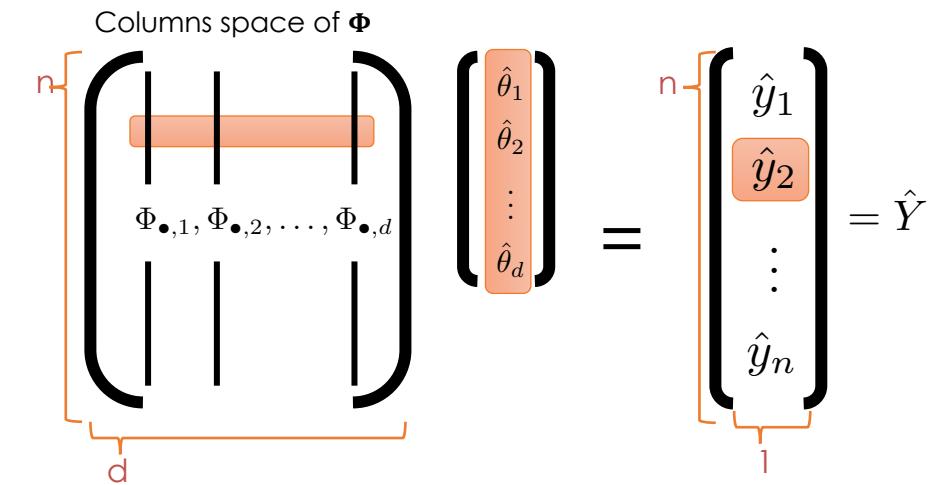
Columns space of Φ

$$\begin{matrix} n \\ \left[\begin{array}{c|c|c|c} & \Phi_{\bullet,1}, \Phi_{\bullet,2}, \dots, \Phi_{\bullet,d} & & \end{array} \right] \\ d \end{matrix} = \begin{pmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ \vdots \\ \hat{\theta}_d \end{pmatrix} = \begin{matrix} n \\ \left[\begin{array}{c|c|c|c} & \hat{y}_1 \\ & \hat{y}_2 \\ & \vdots \\ & \hat{y}_n \end{array} \right] \\ 1 \end{matrix} = \hat{Y}$$



Definition of orthogonality

$$0 = \Phi^T (Y - \Phi\hat{\theta})$$



Derivation

$$0 = \Phi^T (Y - \Phi\hat{\theta})$$

$$0 = \Phi^T Y - \Phi^T \Phi\hat{\theta}$$

$$\Phi^T \Phi\hat{\theta} = \Phi^T Y$$

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T Y$$

“Normal Equation”

The Normal Equation $\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T Y$

$$\hat{\theta} = d^{-1} \begin{pmatrix} \Phi^T \\ \Phi \end{pmatrix}^{-1} \begin{pmatrix} d & \Phi^T & Y \end{pmatrix}$$

Note: For inverse to exist Φ needs to be full column rank.

→ cannot have co-linear features

This can be addressed by adding regularization ...

**In practice we will use regression software
(e.g., scikit-learn) to estimate θ**

Least Squares Regression in Practice

- Use optimized software packages
 - Address numerical issues with matrix inversion
- Incorporate some form of regularization
 - Address issues of collinearity
 - Produce more robust models
- We will be using scikit-learn:
 - http://scikit-learn.org/stable/modules/linear_model.html
 - See Homework 6 for details!

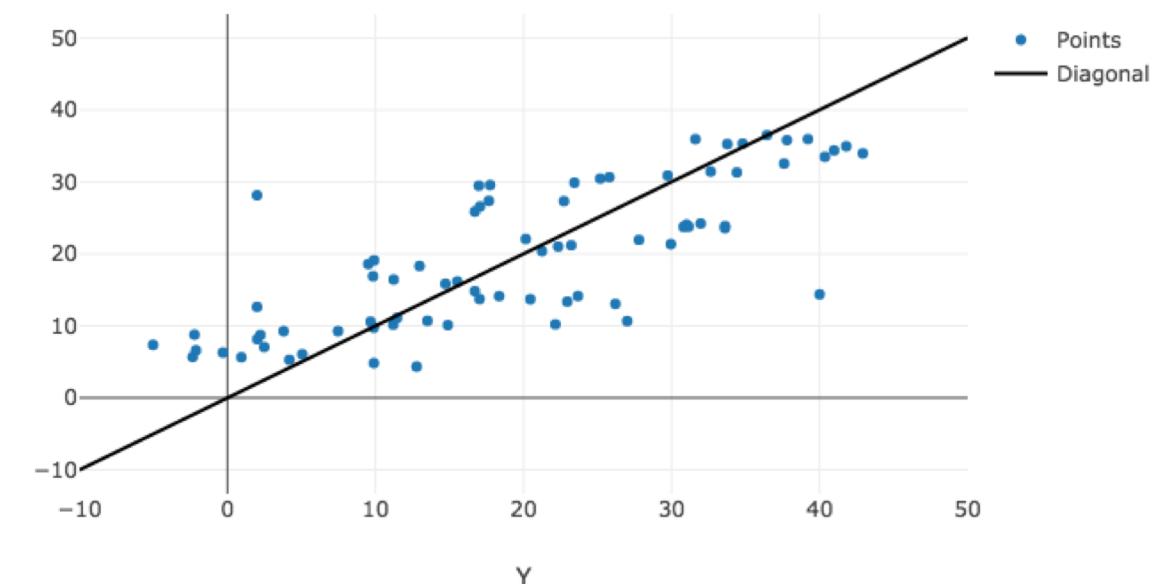
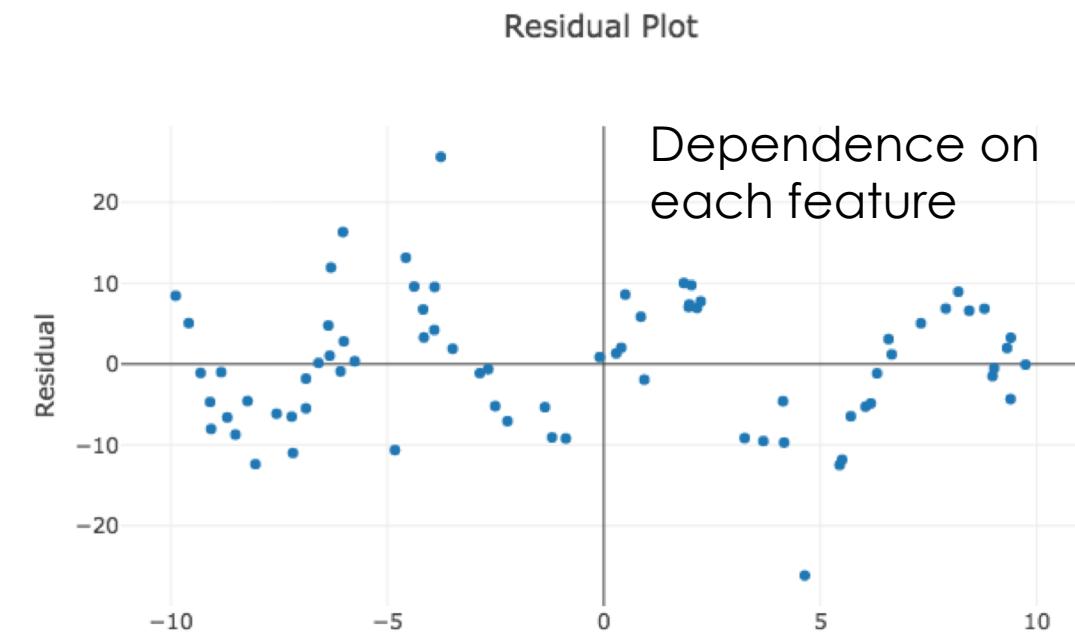
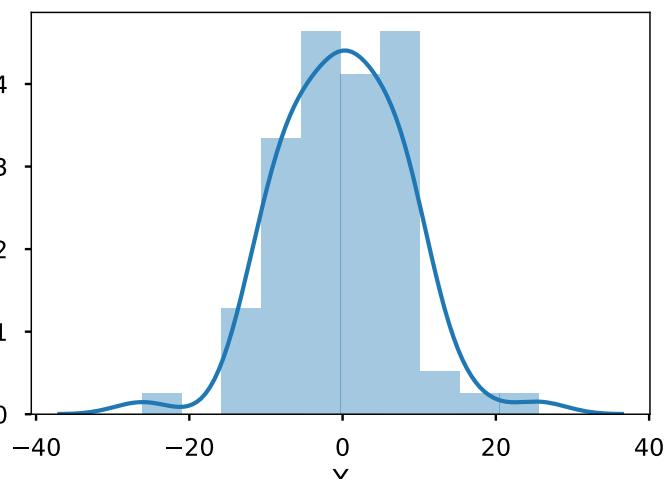
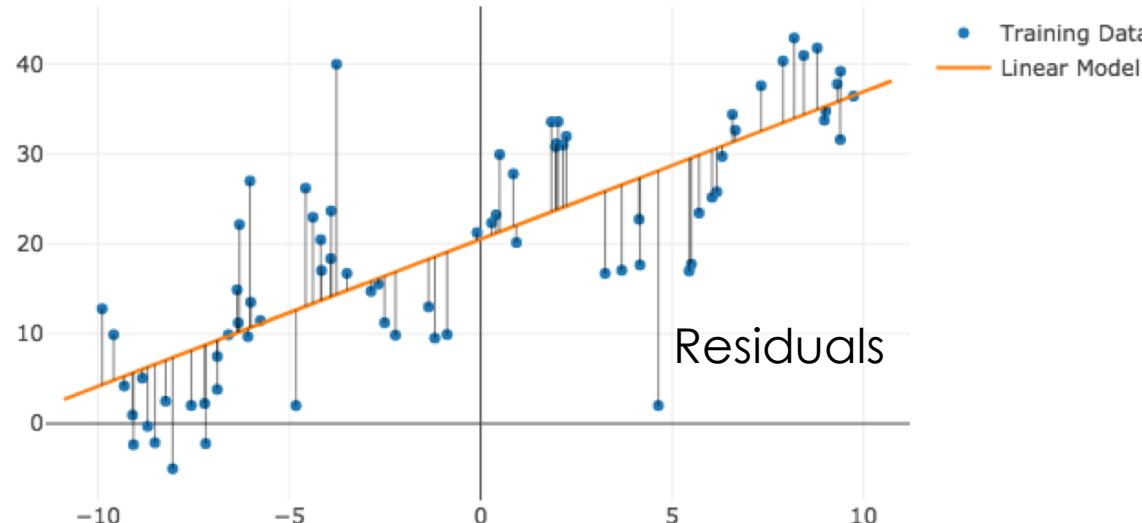
Scikit Learn Models

- Scikit Learn has a wide range of models
- Many of the models follow a common pattern:

Ordinary Least Squares Regression

```
from sklearn import linear_model  
  
f = linear_model.LinearRegression(fit_intercept=True)  
  
f.fit(train_data[['X']], train_data['Y'])  
  
Yhat = f.predict(test_data[['X']])
```

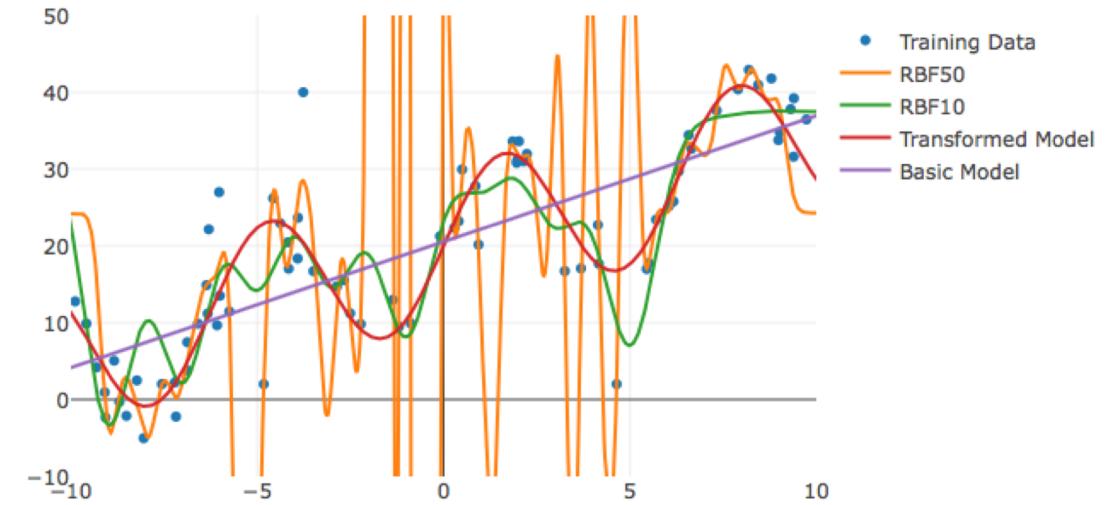
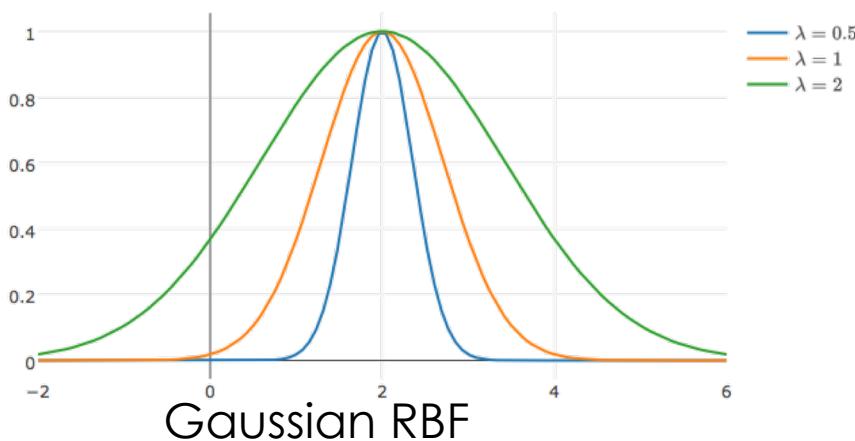
Diagnosing Fit → The Residuals



Notebook Demo

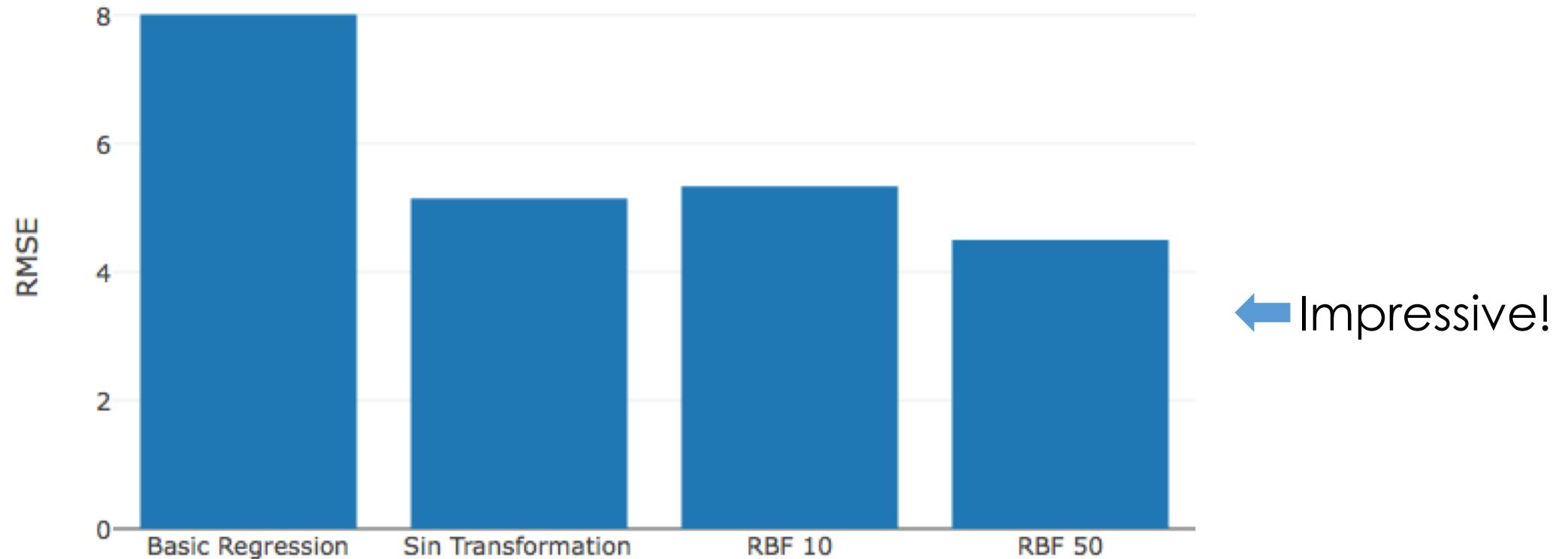
- **Generic Features:** increase model expressivity
- **Gaussian Radial Basis Functions:**

$$\phi_{\lambda_i, \mu_i}(x) = \exp\left(-\frac{\|x - \mu_i\|_2^2}{\lambda_i}\right)$$

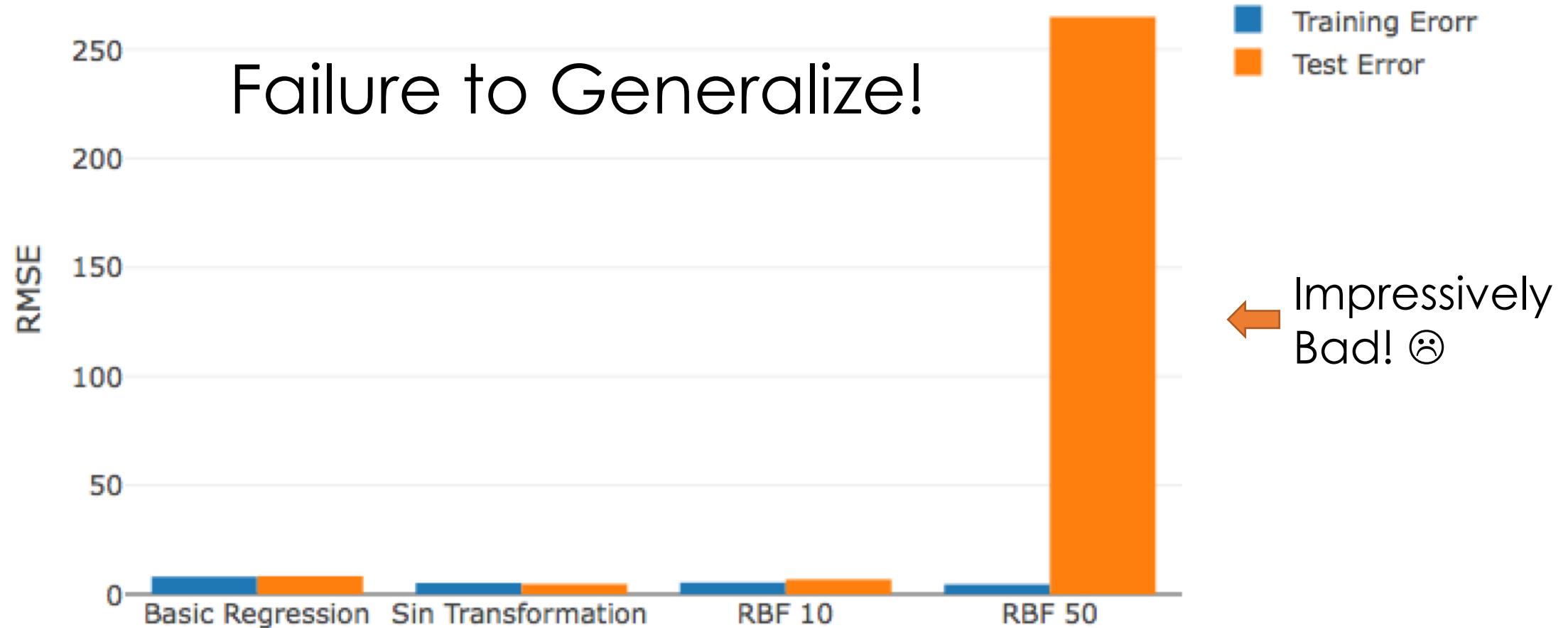


Training Error

Loss Comparison

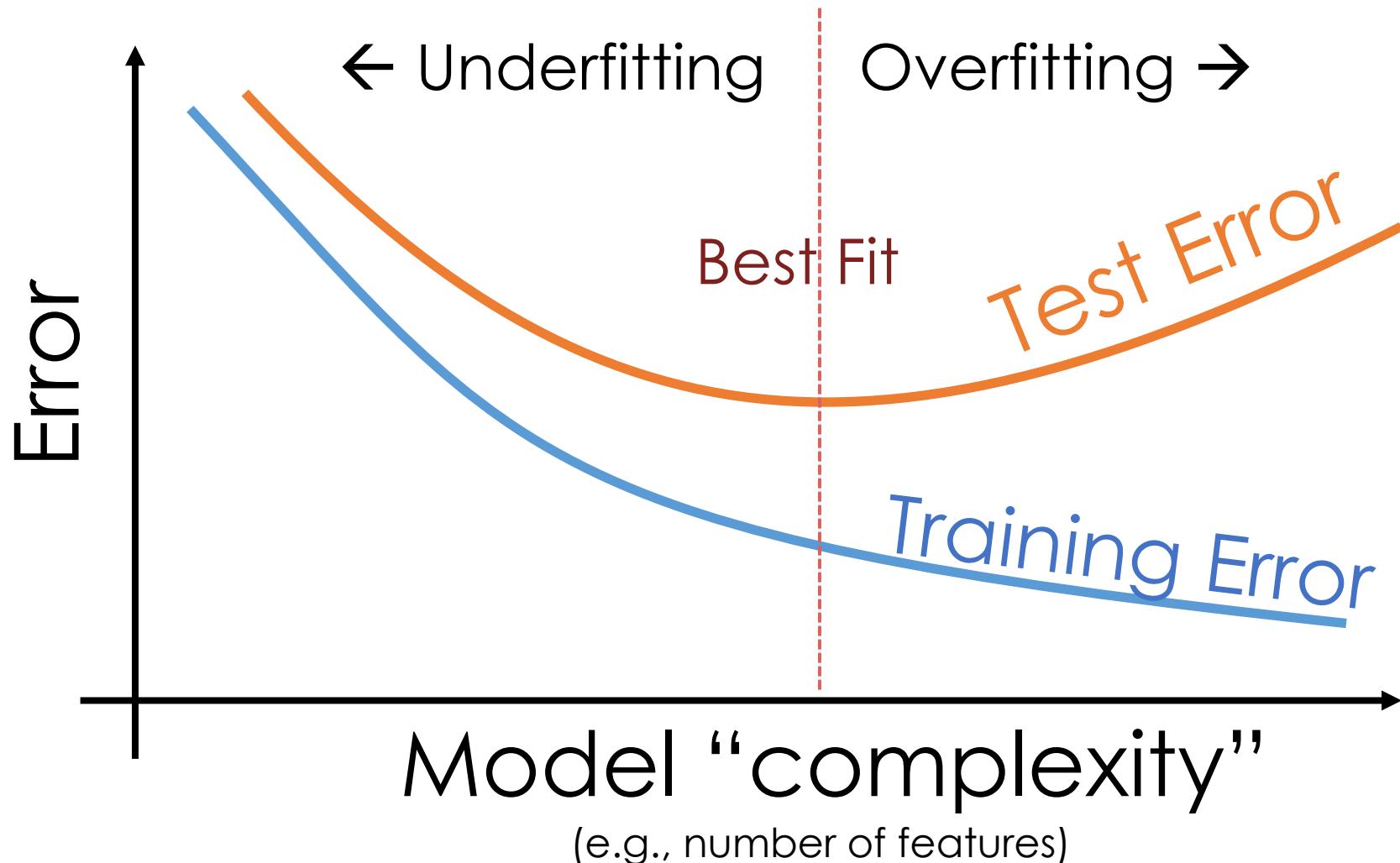


Training vs Test Error



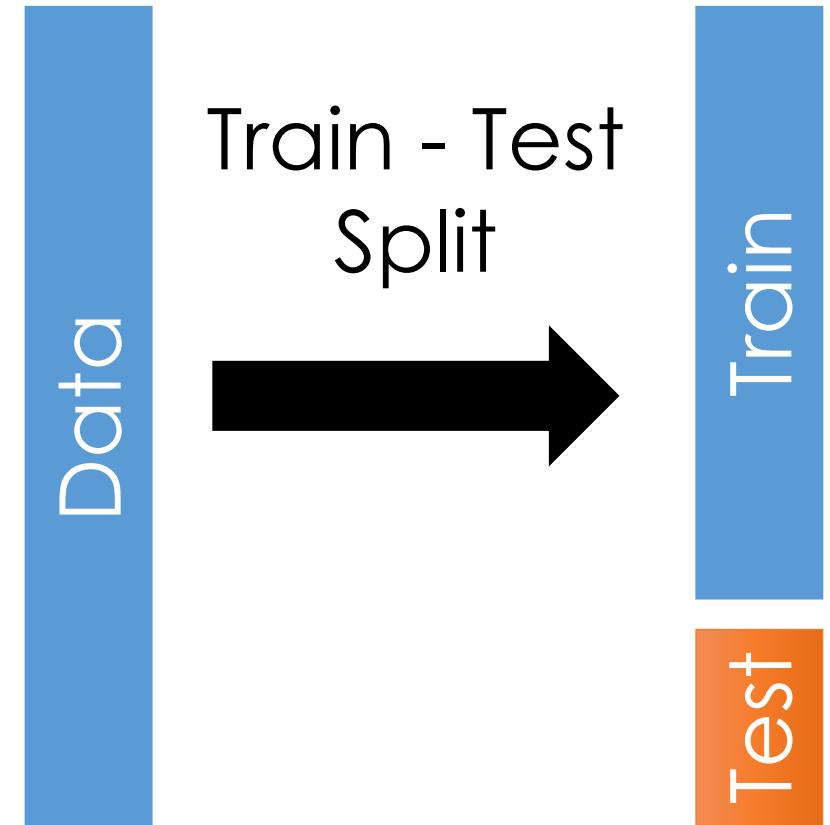
Training vs Test Error

Training error typically under estimates test error.



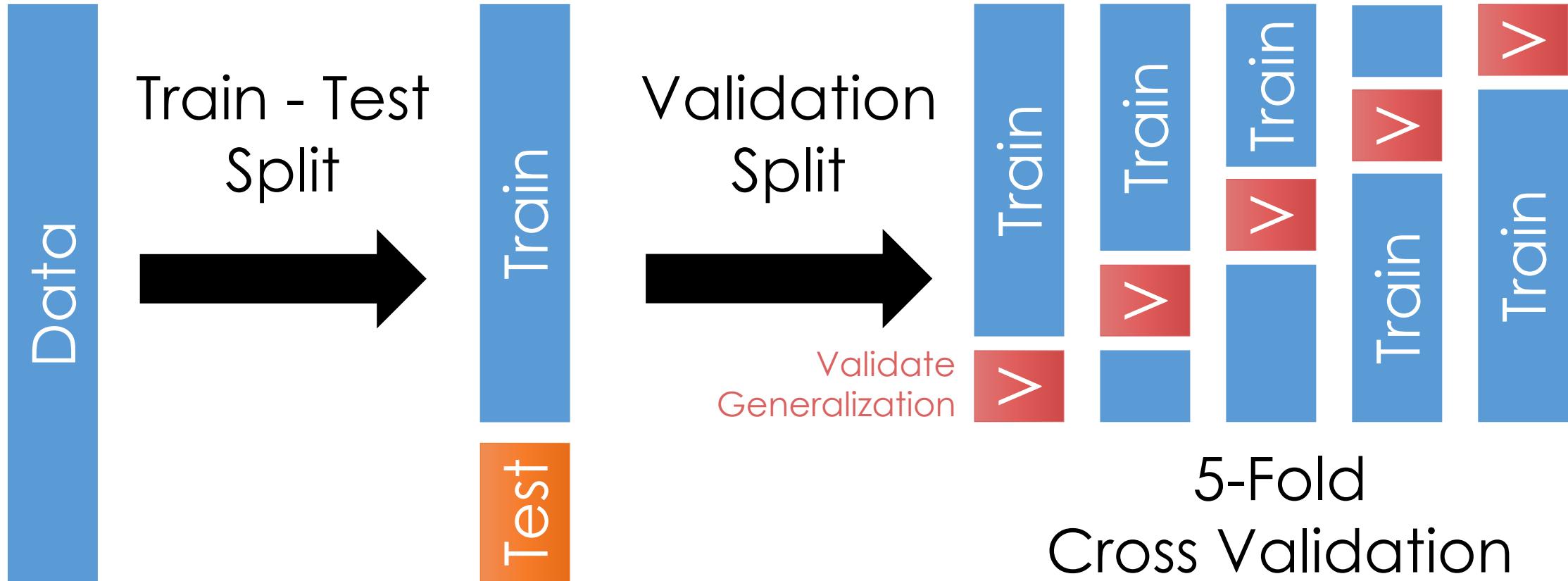
Generalization: The Train-Test Split

- **Training Data:** used to fit model
- **Test Data:** check generalization error
- How to split?
 - Randomly, Temporally, Geo...
 - Depends on application (usually randomly)
- What size? (90%-10%)
 - Larger training set → more complex models
 - Larger test set → better estimate of generalization error
 - Typically between 75%-25% and 90%-10%



You can only use the test dataset once after deciding on the model.

Generalization: Validation Split



Cross validation **simulates multiple train test-splits** on the training data.

Recipe for Successful Generalization

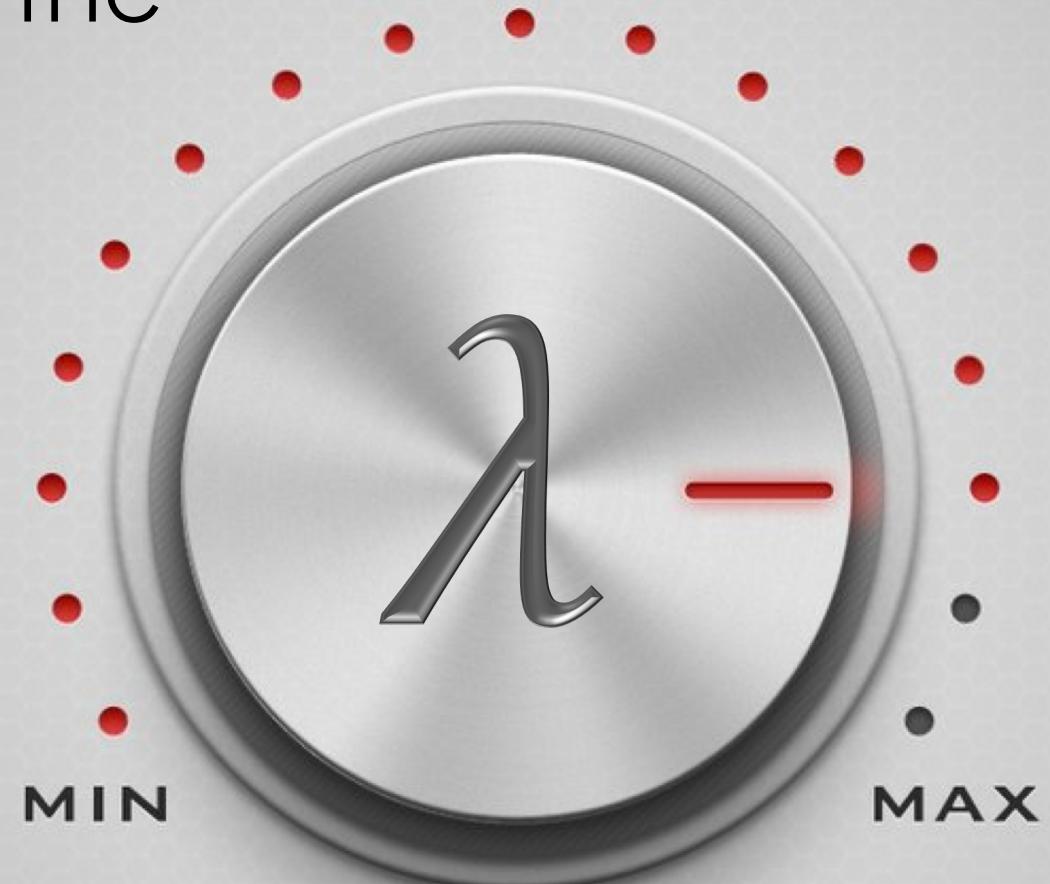
1. Split your data into **training** and **test** sets (90%, 10%)
2. Use **only the training data** when designing, training, and tuning the model
 - Use **cross validation** to test generalization during this phase
 - **Do not look at the test data**
3. Commit to your final model and train once more using **only the training data**.
4. Test the final model using the **test data**. If accuracy is not acceptable return to (2). (Get more *test data* if possible.)
5. Train **on all available data** and ship it!



Returning to Regularization

Regularization

Parametrically Controlling the
Model Complexity

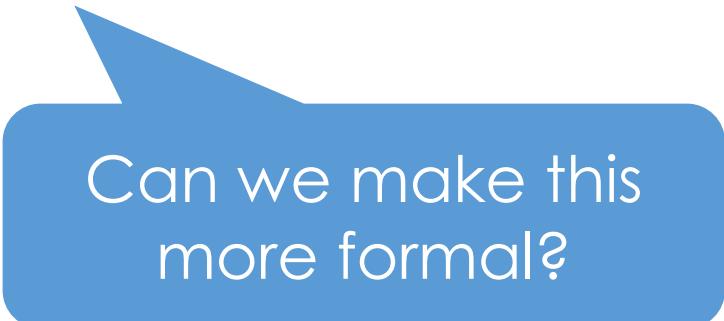


Basic Idea

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_{\theta}(x_i))$$

Such that:

f_{θ} is not too “complicated”



Can we make this
more formal?

Basic Idea

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_{\theta}(x_i))$$

Such that:

$$\text{Complexity}(f_{\theta}) \leq \beta$$

Regularization
Parameter

How do we
define this?

Idealized Notion of Complexity

$$\text{Complexity}(f_\theta) \leq \beta$$

- Focus on complexity of **linear models**:
 - Number and kinds of features
- Ideal definition:

$$\text{Complexity}(f_\theta) = \sum_{j=1}^d \mathbb{I}[\theta_j \neq 0]$$

Number of
non-zero
parameters

- Why?

Ideal “Regularization”

Find the best value of θ which uses fewer than β features.

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_{\theta}(x_i))$$

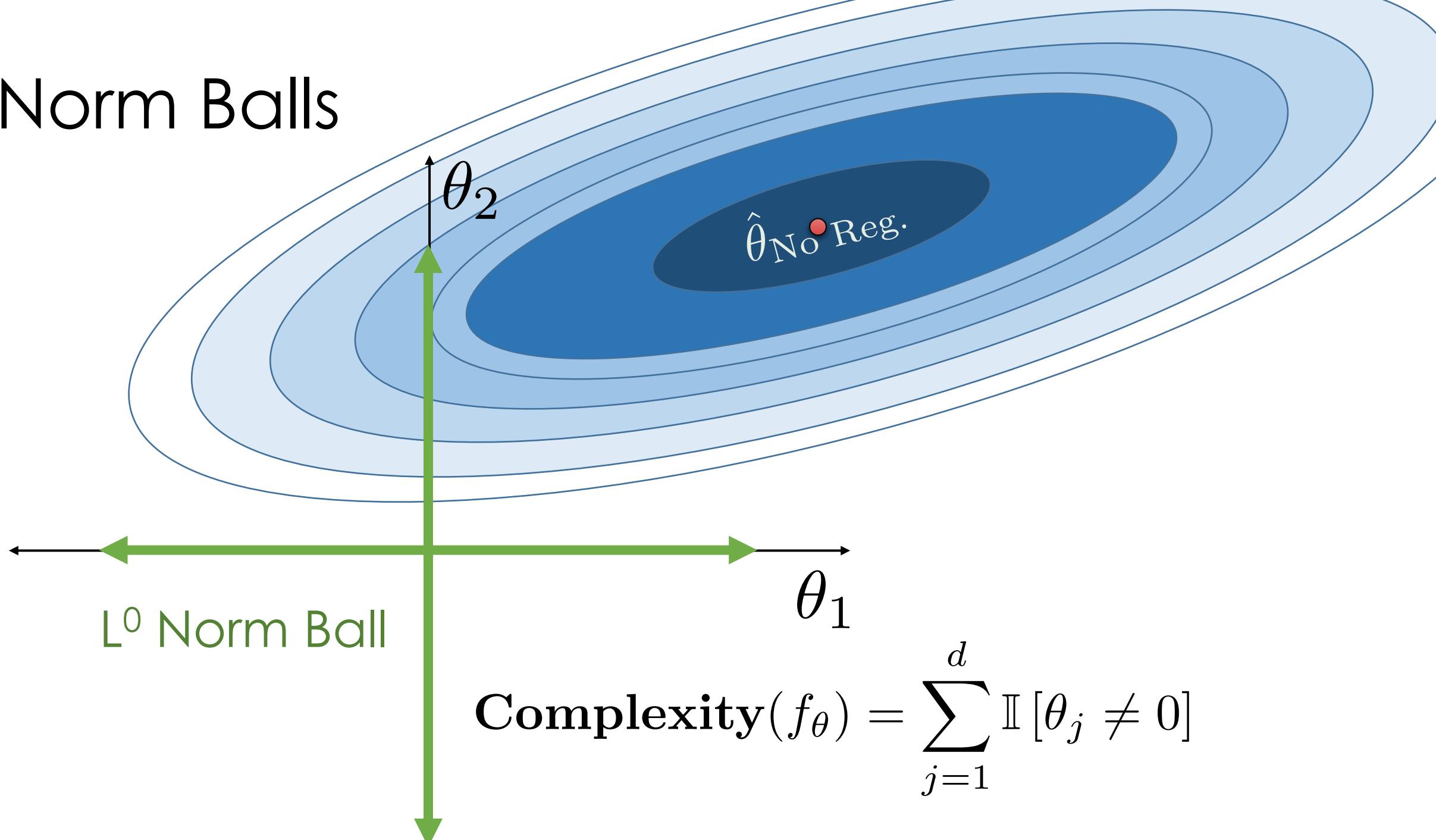
Such that:

Need an approximation!

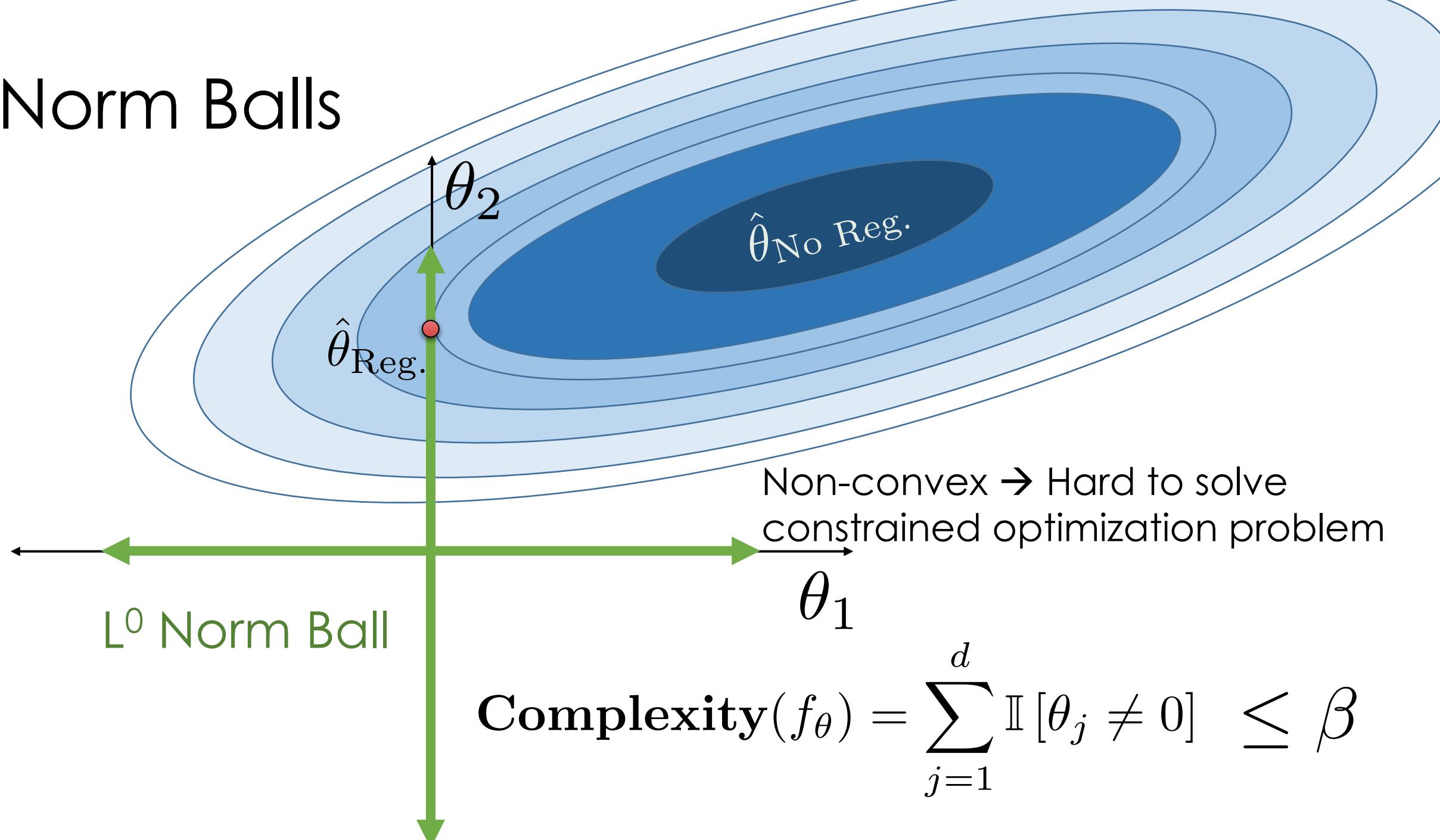
$$\text{Complexity}(f_{\theta}) = \sum_{j=1}^d \mathbb{I}[\theta_j \neq 0] \leq \beta$$

Combinatorial search problem \rightarrow NP-hard to solve in general.

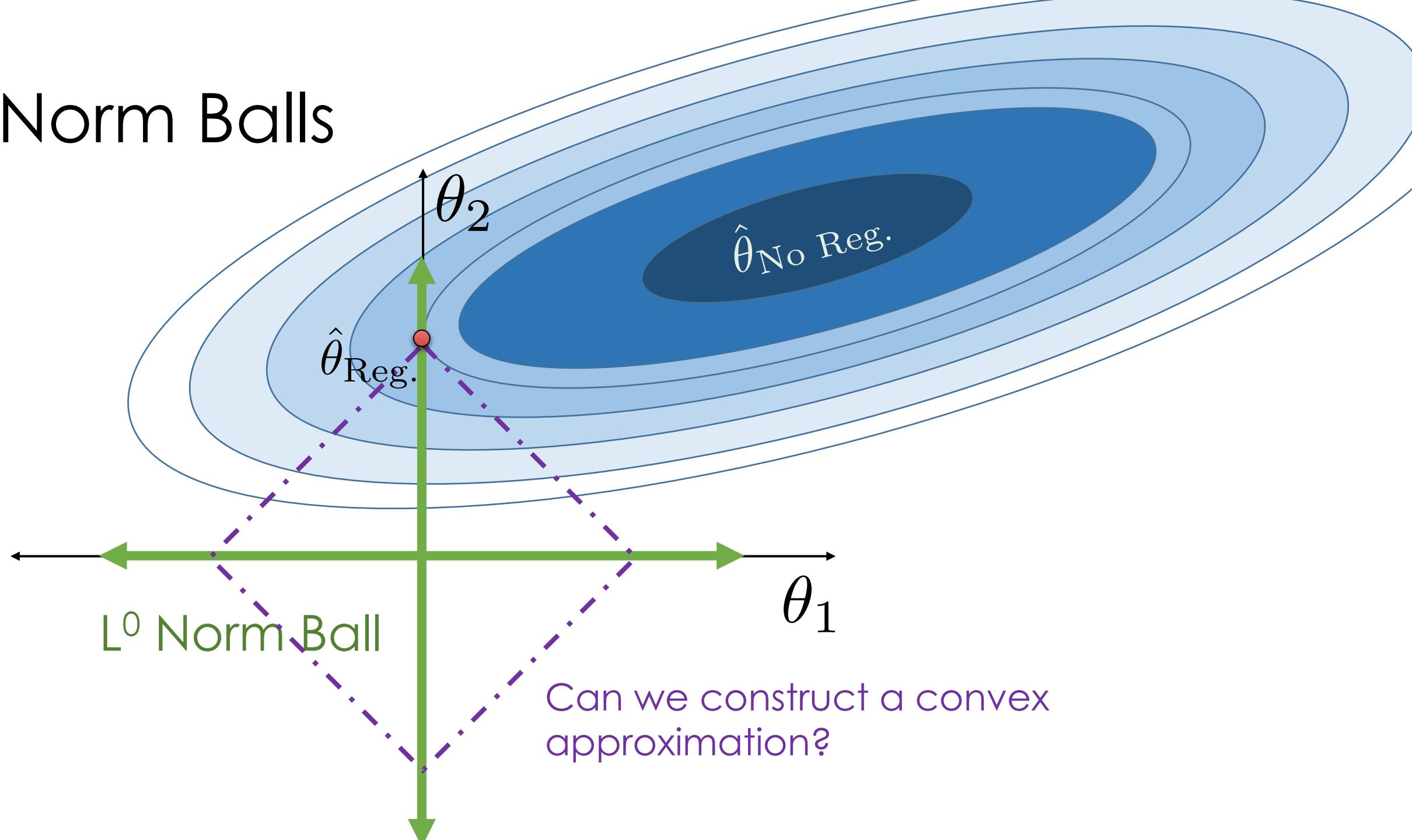
Norm Balls



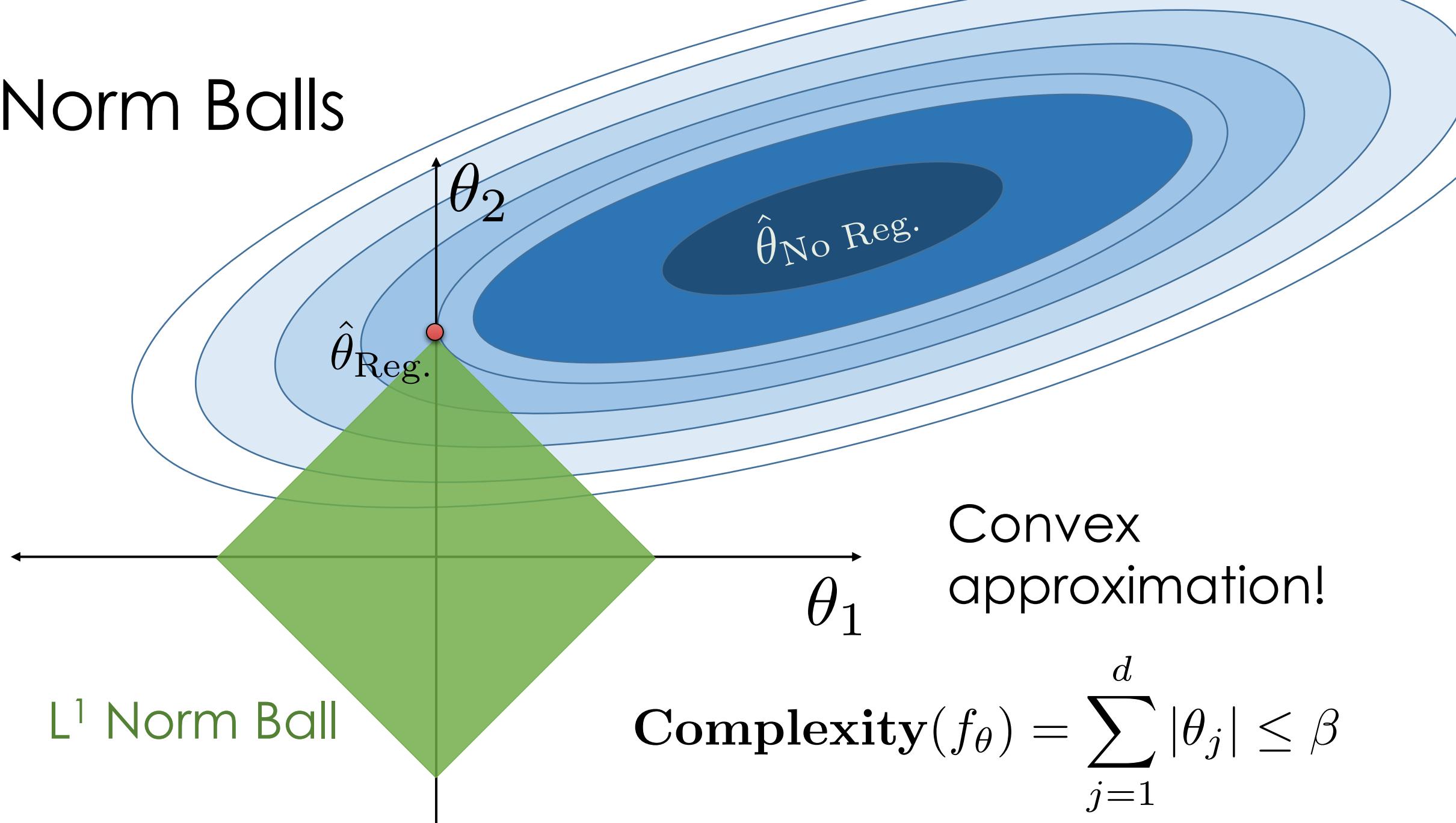
Norm Balls



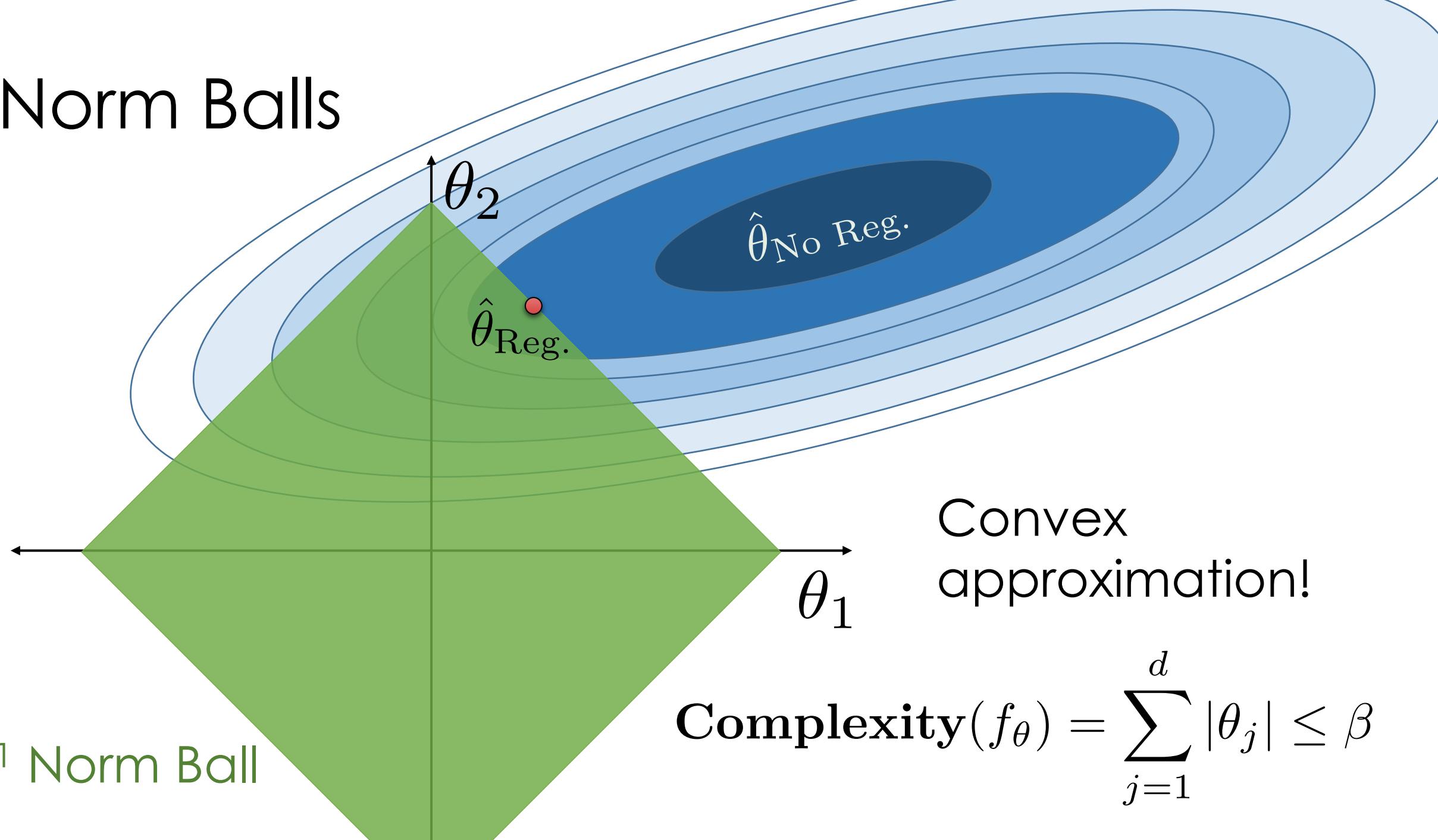
Norm Balls



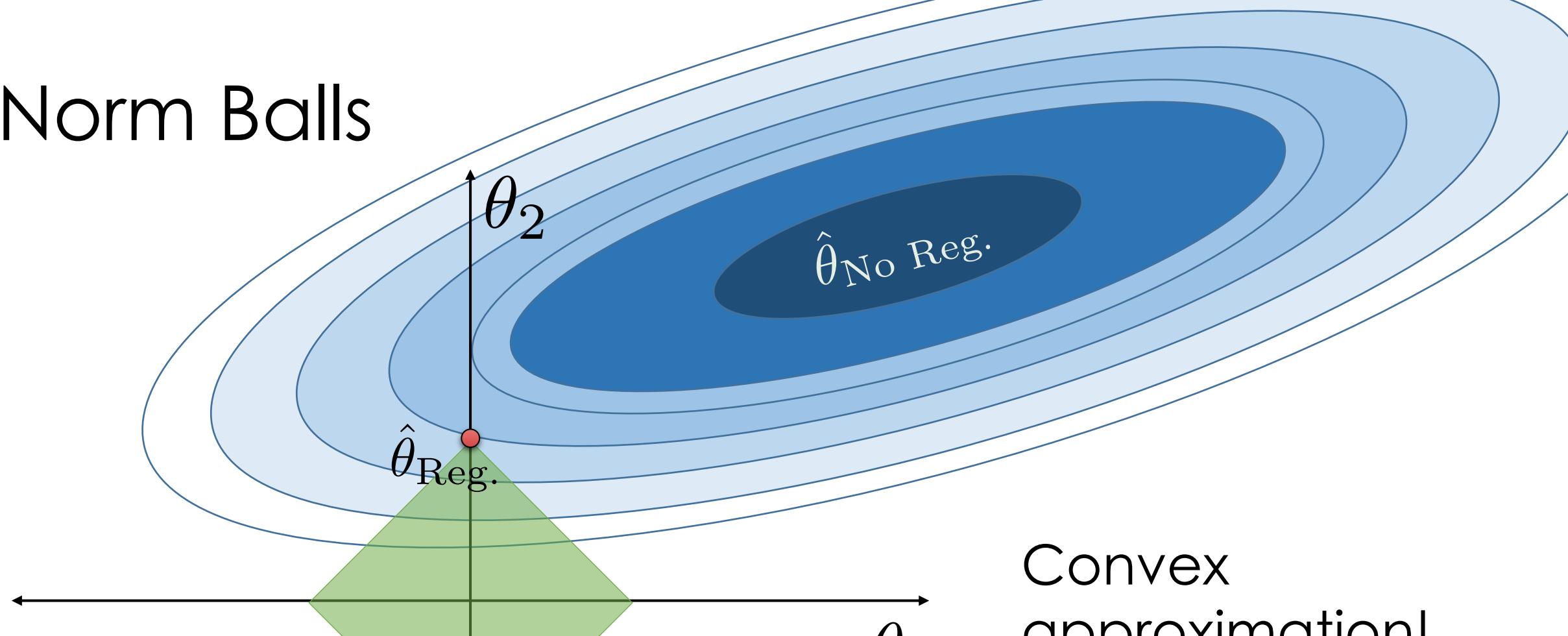
Norm Balls



Norm Balls



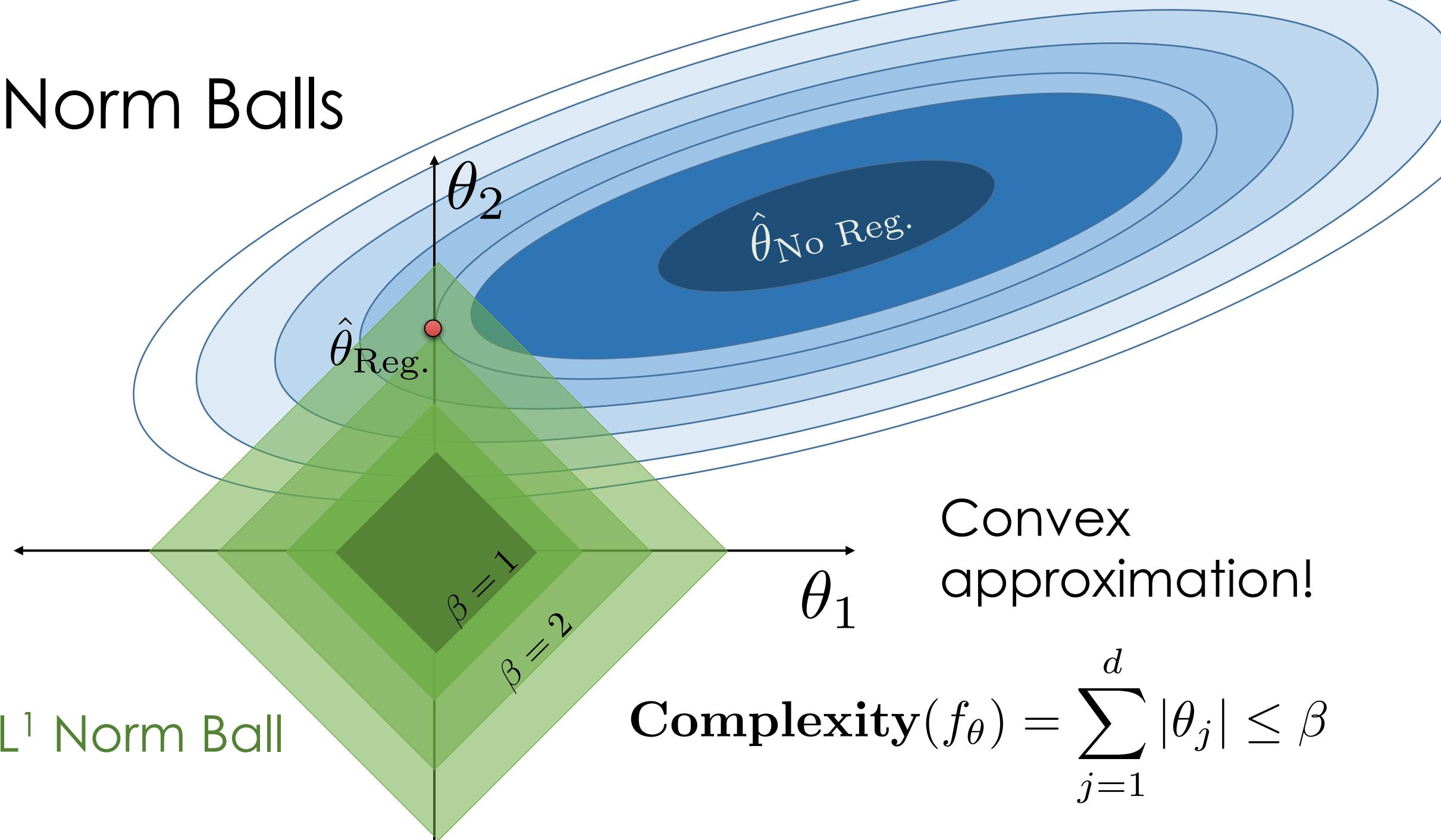
Norm Balls



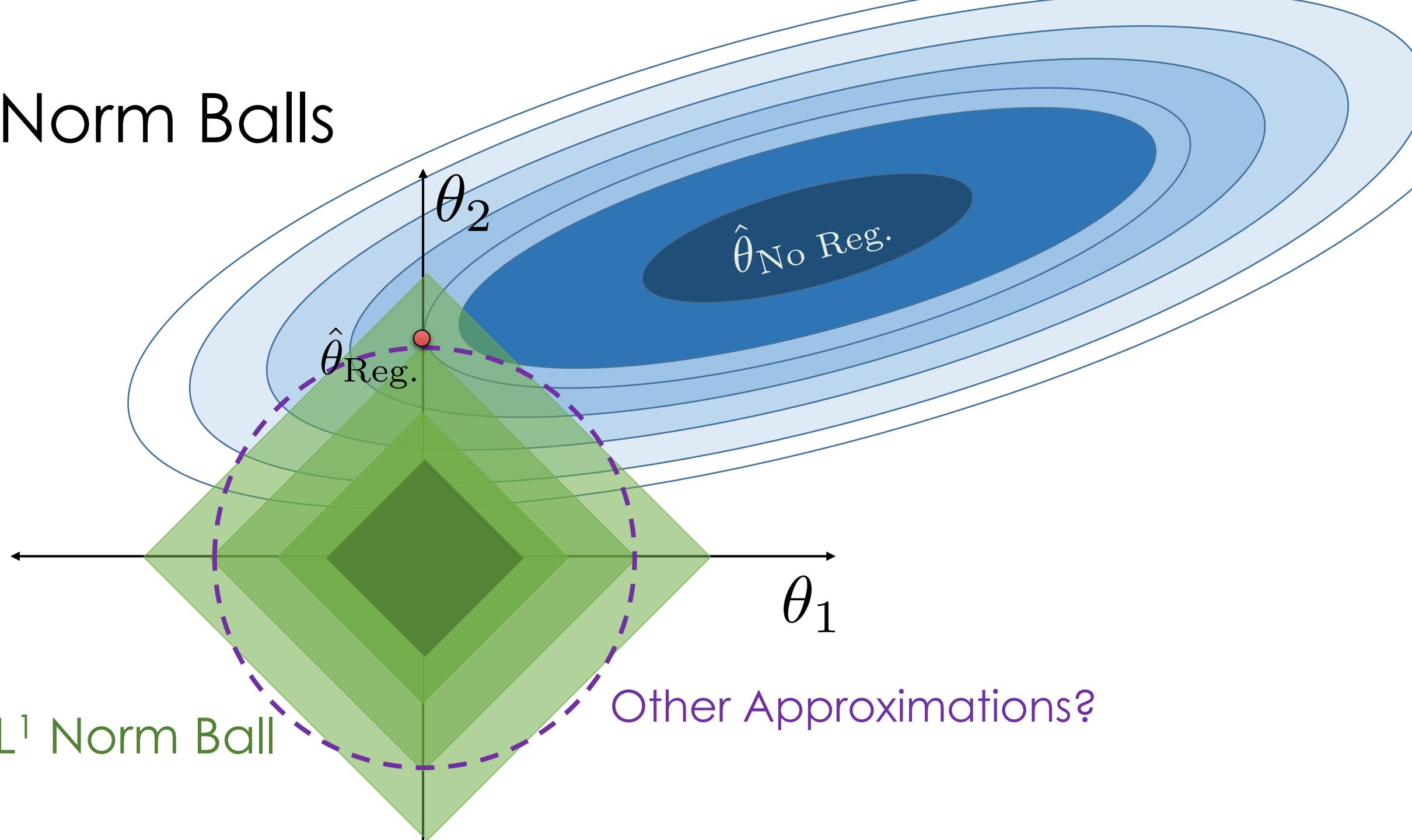
L^1 Norm Ball

Complexity(f_θ) = $\sum_{j=1}^d |\theta_j| \leq \beta$

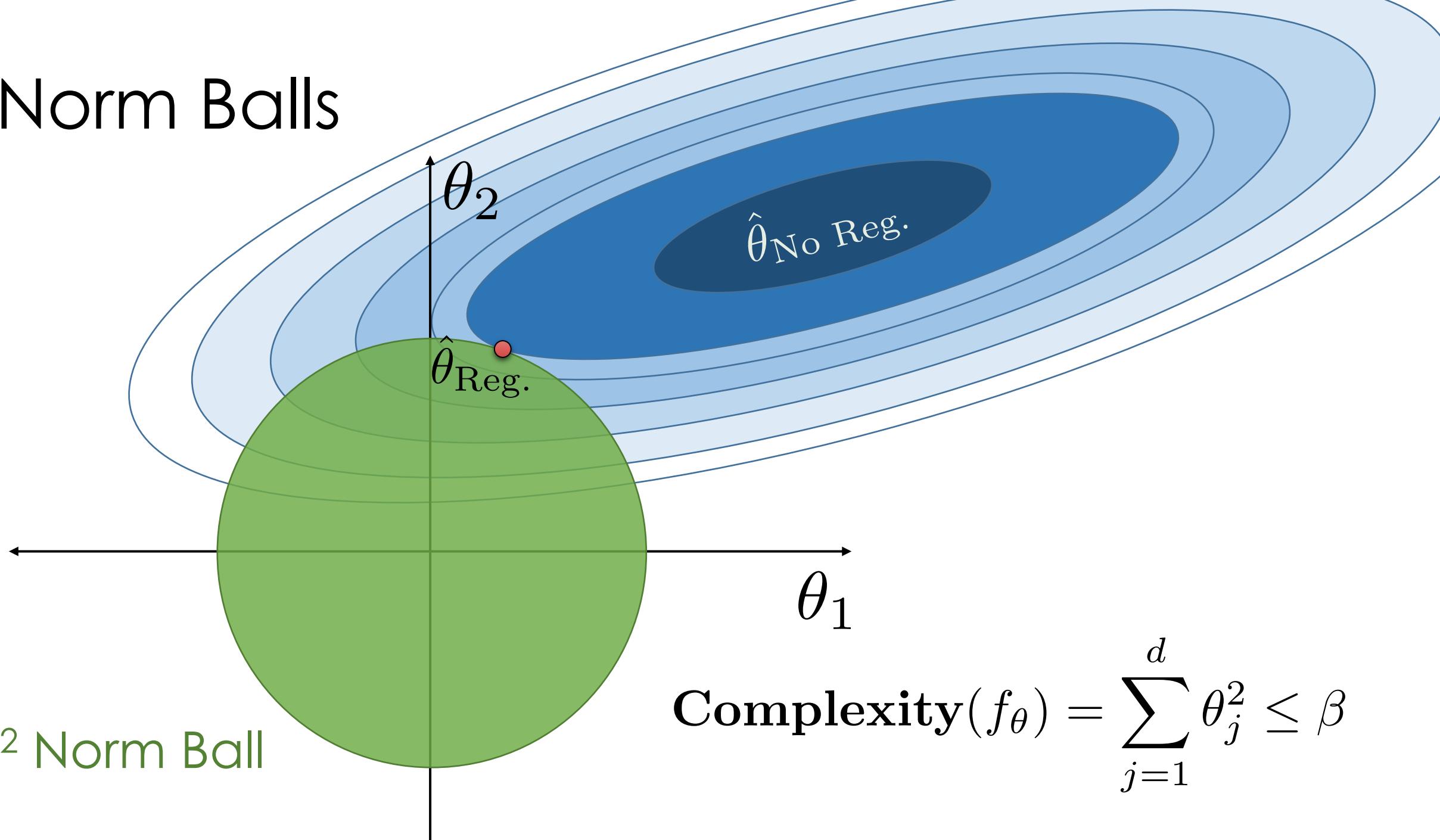
Norm Balls



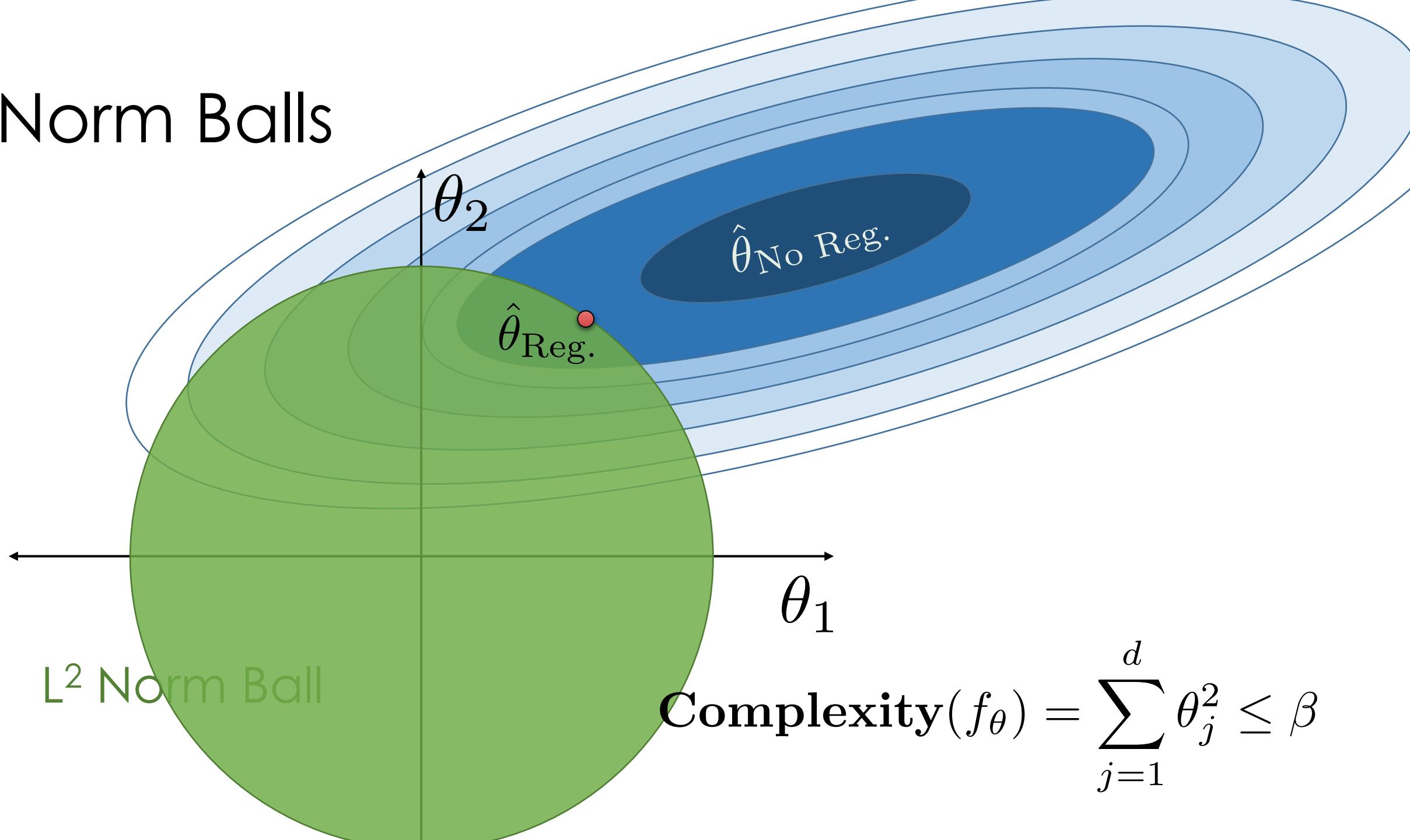
Norm Balls



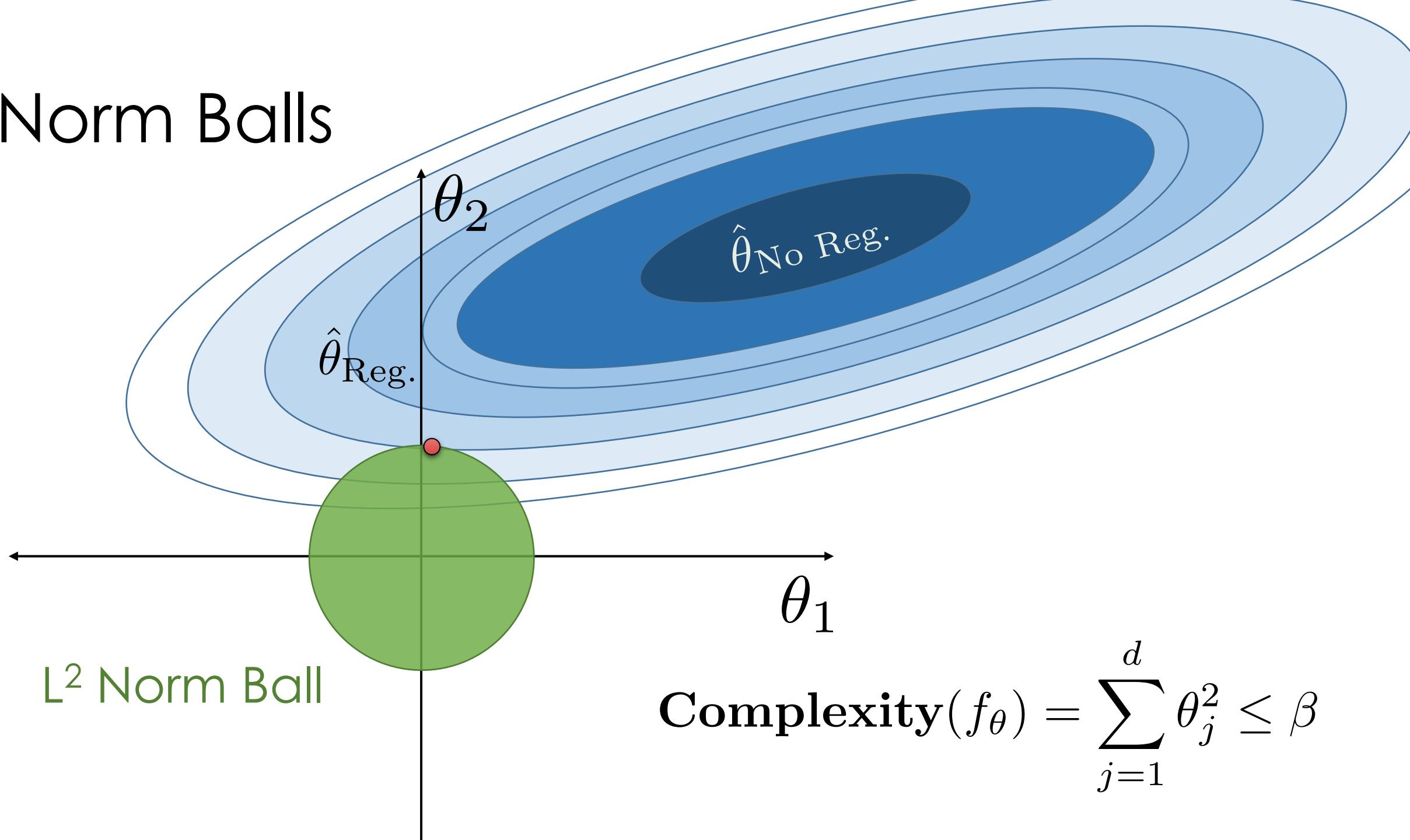
Norm Balls



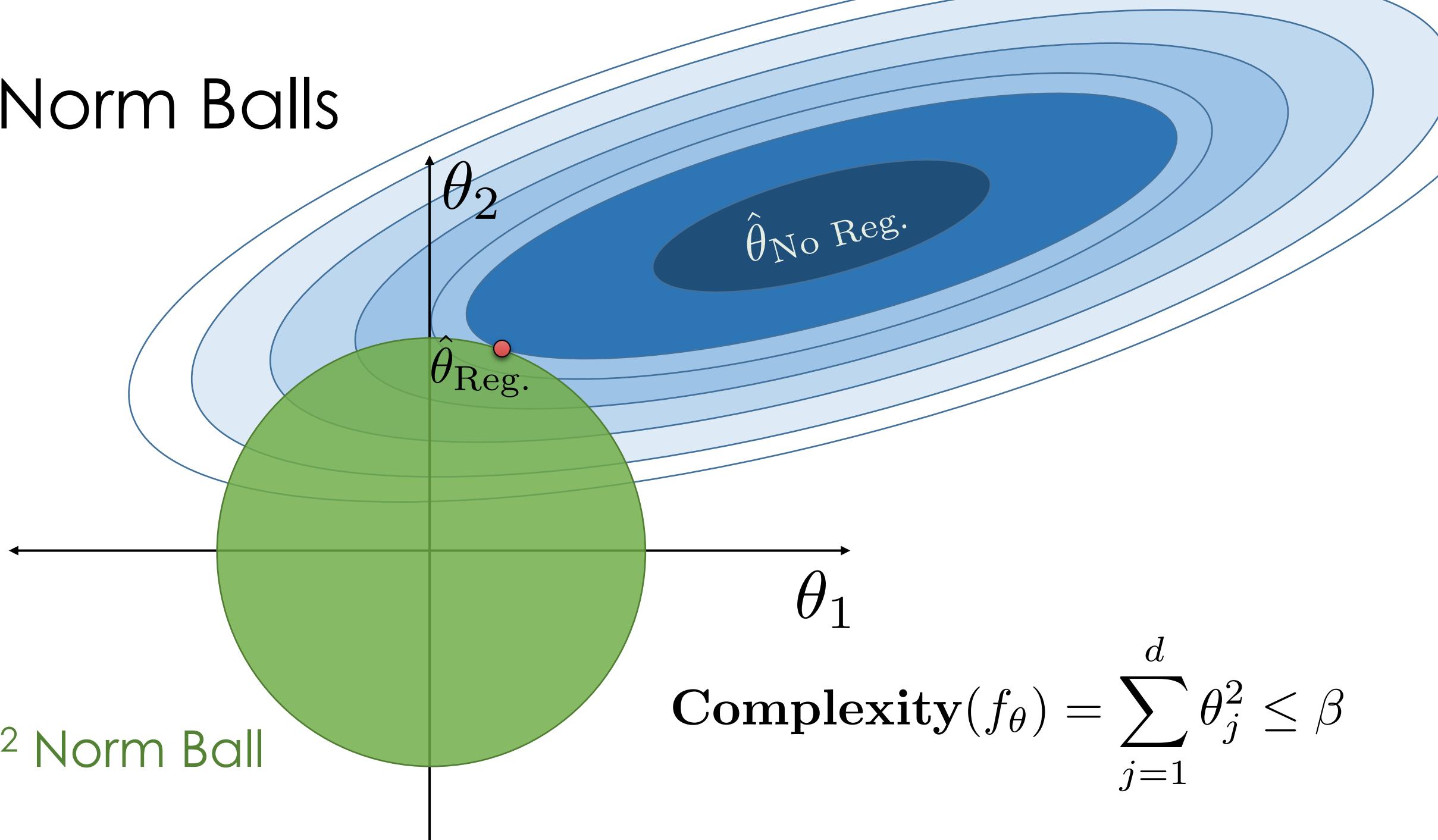
Norm Balls



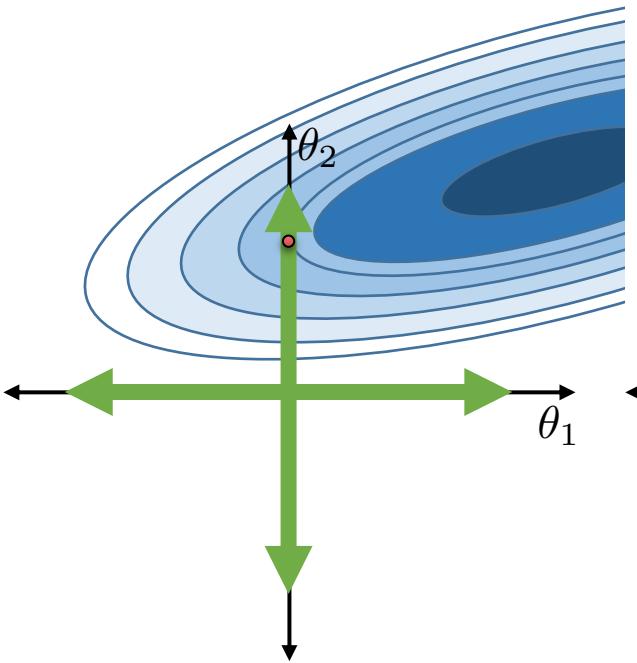
Norm Balls



Norm Balls

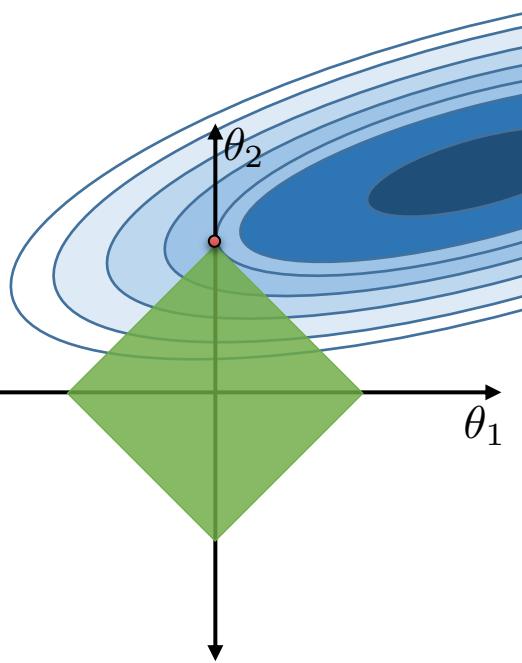


L^0 Norm Ball



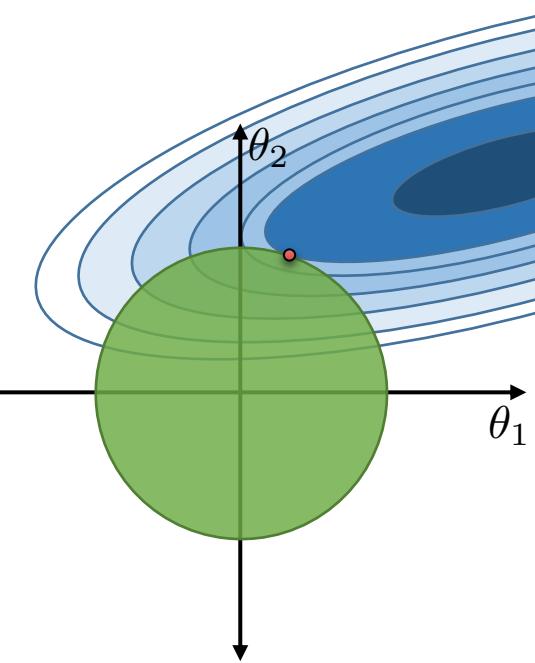
Ideal for Feature Selection
but combinatorically difficult to optimize

L^1 Norm Ball



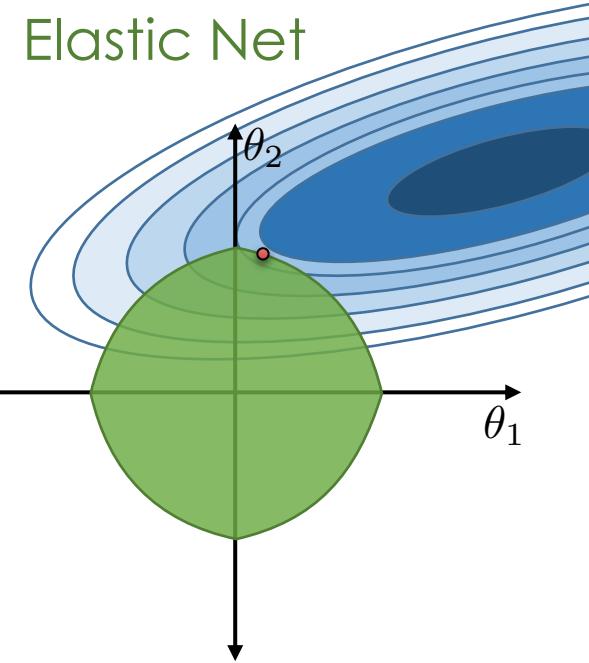
Encourages Sparse Solutions
Convex!

L^2 Norm Ball



Spreads weight over features (**robust**)
does not encourage sparsity

$L^1 + L^2$ Norm Elastic Net



Compromise
Need to tune two regularization parameters

Generic Regularization (Constrained)

- Defining **Complexity**(f_θ) = $R(\theta)$

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_\theta(x_i))$$

Such that: $R(\theta) \leq \beta$

- There is an equivalent unconstrained formulation (obtained by Lagrangian duality)

Generic Regularization (Constrained)

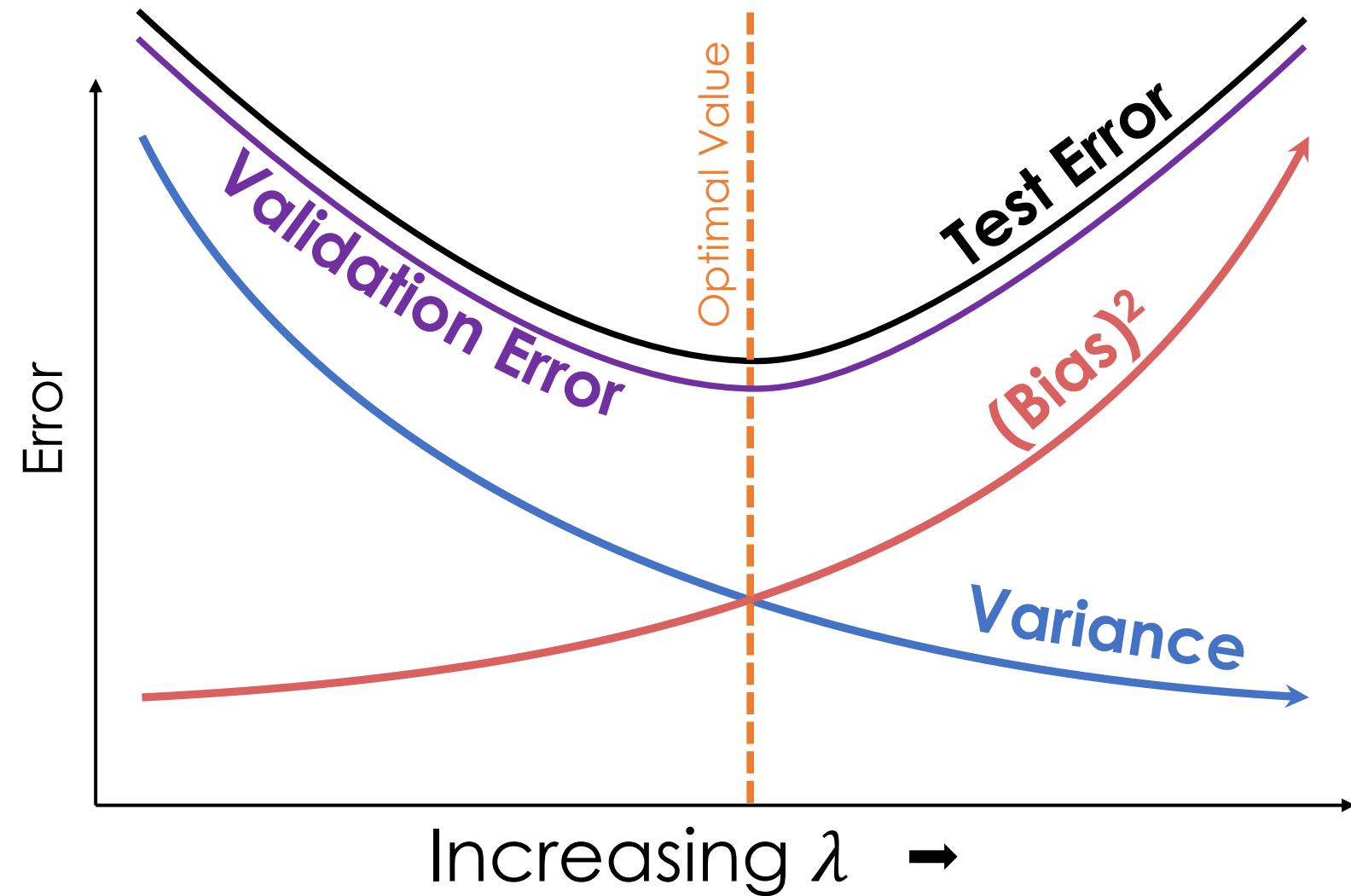
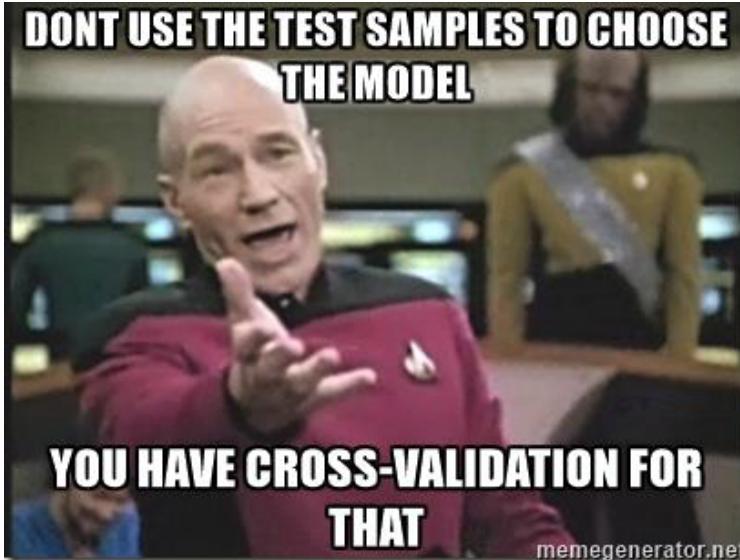
- Defining **Complexity**(f_θ) = $R(\theta)$

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f_\theta(x_i)) + \lambda R(\theta)$$

Regularization
Parameter

- There is an equivalent unconstrained formulation (obtained by Lagrangian duality)

Determining the Optimal λ



- Value of λ determines bias-variance tradeoff
 - Larger values \rightarrow more regularization \rightarrow more bias \rightarrow less variance
- Determined through cross validation

Using Scikit-Learn for Regularized Regression

```
import sklearn.linear_model
```

- Regularization parameter $\alpha = 1/\lambda$
 - larger $\alpha \rightarrow$ less regularization \rightarrow greater complexity \rightarrow overfitting
- Lasso Regression (L1)
 - `linear_model.Lasso(alpha=3.0)`
 - `linear_model.LassoCV()` automatically picks α by cross-validation
- Ridge Regression (L2)
 - `linear_model.Ridge(alpha=3.0)`
 - `linear_model.RidgeCV()` automatically selects α by cross-validation
- Elastic Net (L1 + L2)
 - `linear_model.ElasticNet(alpha=3.0, l1_ratio = 2.0)`
 - `linear_model.ElasticNetCV()` automatically picks α by cross-validation

Standardization and the Intercept Term

Height = θ_1 age_in_seconds + θ_2 weight_in_tons

Small

Large

➤ Regularization penalized dimensions equally

➤ **Standardization**

- Ensure that each dimensions has the same scale
- centered around zero

Standardization

For each dimension k :

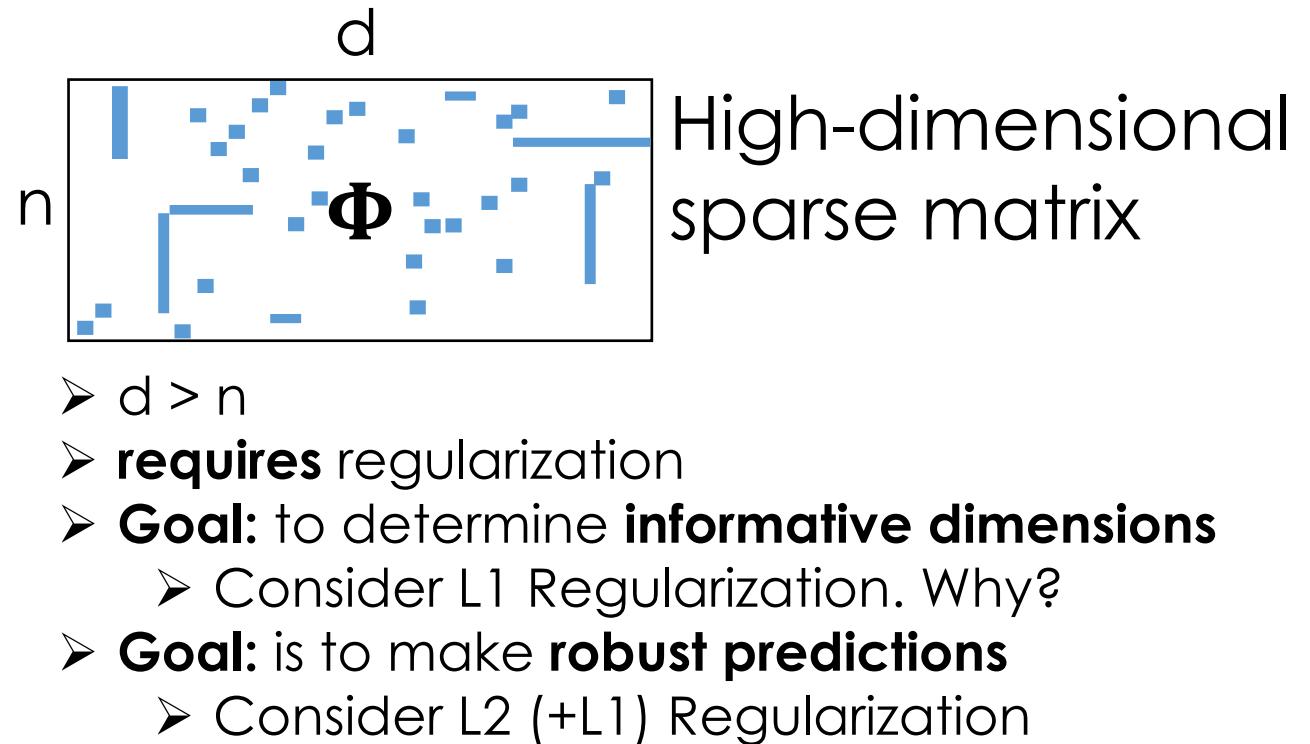
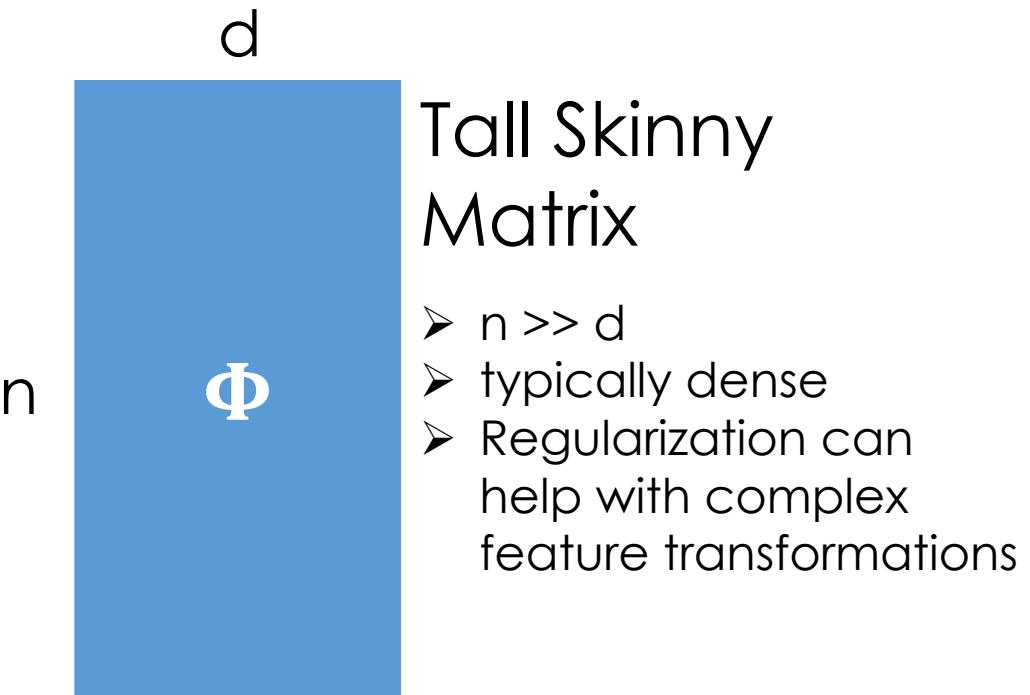
$$z_k = \frac{x_k - \mu_k}{\sigma_k}$$

➤ **Intercept Terms**

- Typically don't regularize intercept term
- Center y values (e.g., subtract mean)

Regularization and High-Dimensional Data

Regularization is often used with high-dimensional data



Matching!

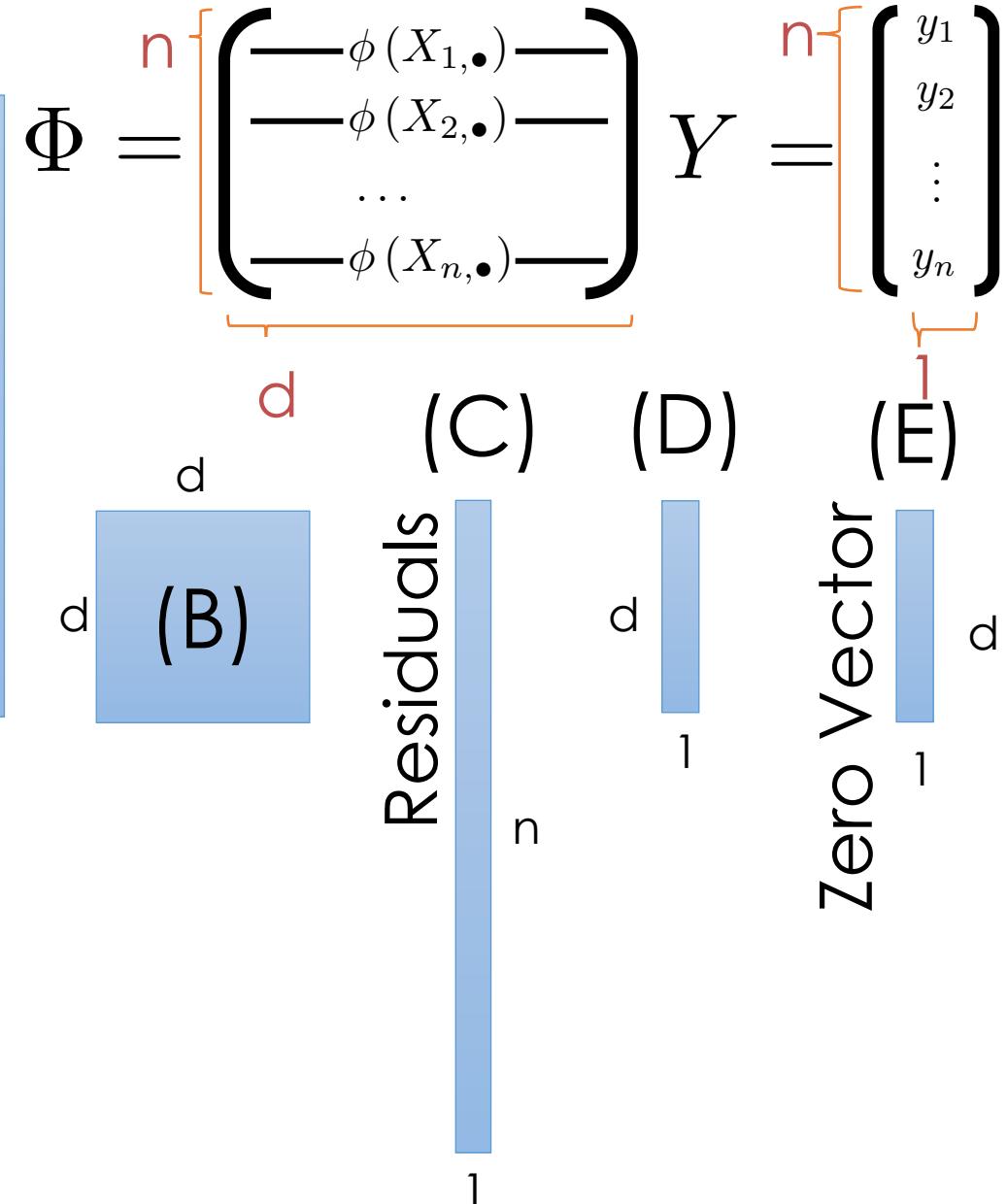
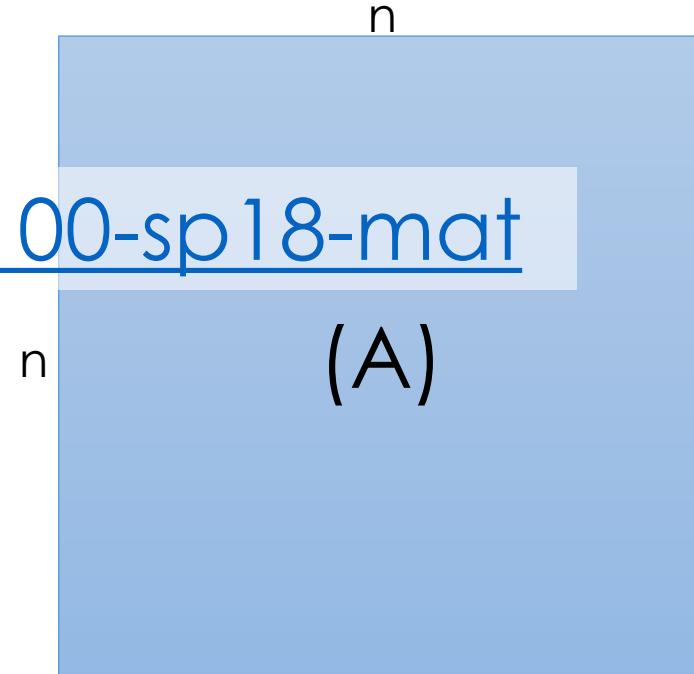
<http://bit.ly/ds100-sp18-mat>

1) $\Phi^T \Phi$

2) $\Phi^T Y$

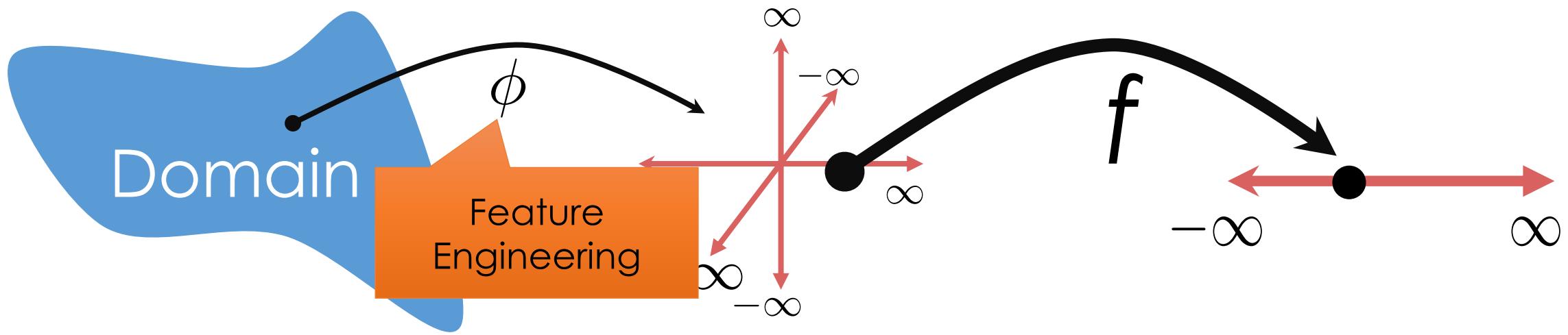
3) $(I - \Phi(\Phi^T \Phi)^{-1} \Phi^T) Y$

4) $\Phi^T (I - \Phi(\Phi^T \Phi)^{-1} \Phi^T) Y$



Classification

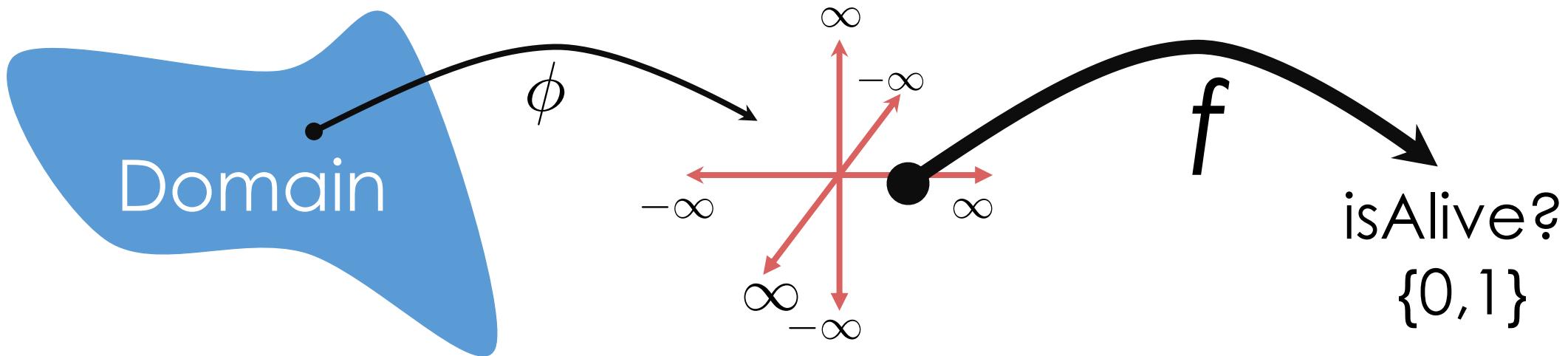
Recap: Least Squares Regression



$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n (y_i - f_{\theta}(x_i))^2 + \lambda R(\theta)$$

Regularization

Classification

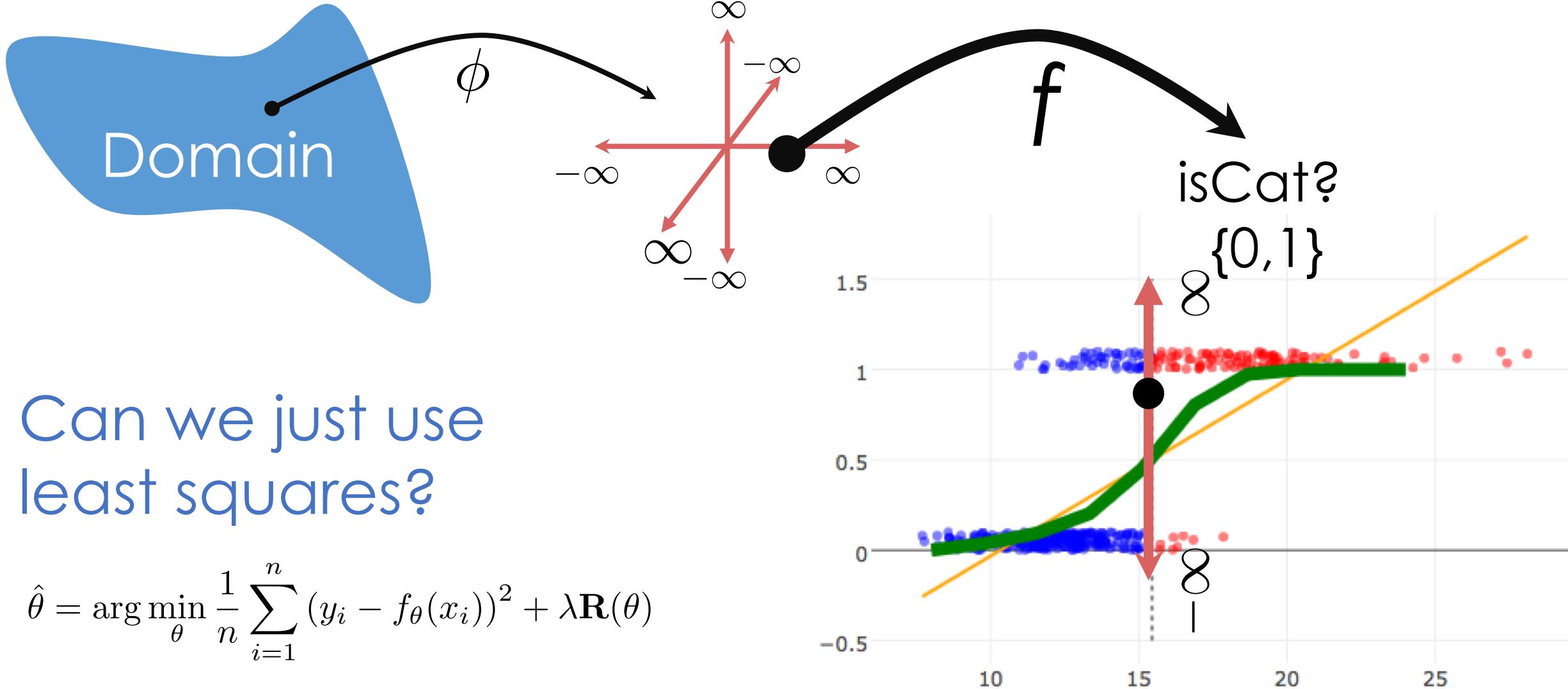


Can we just use least squares?

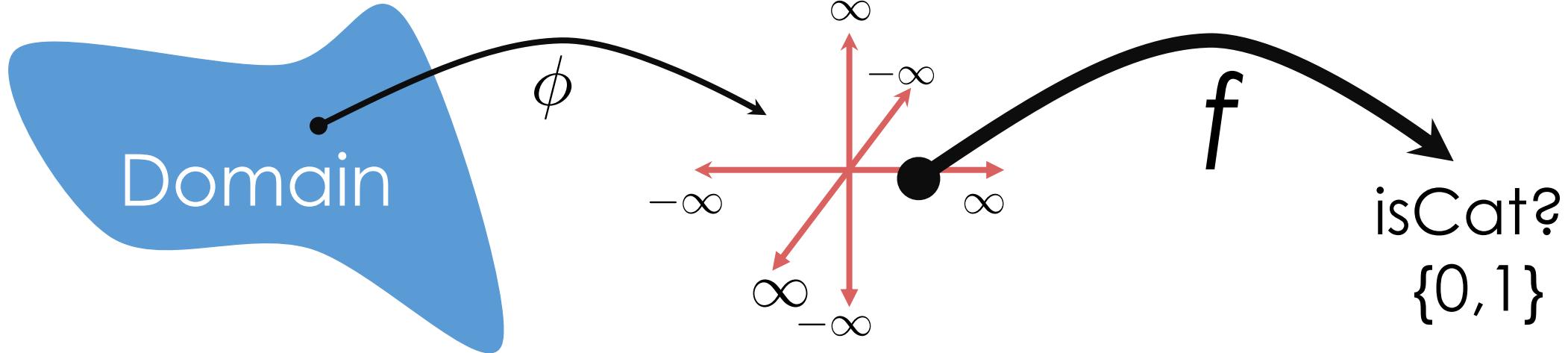
$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n (y_i - f_{\theta}(x_i))^2 + \lambda \mathbf{R}(\theta)$$

Python Demo

Classification



Classification



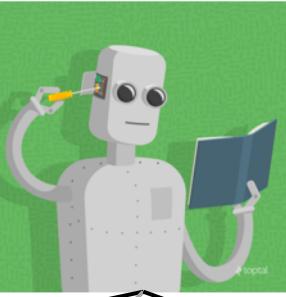
Can we just use least squares?

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n (y_i - f_{\theta}(x_i))^2 + \lambda R(\theta)$$

- Yes ... (can be easy to implement)
- Needs to be modified (e.g., **Don't use Least Squares for Classification**)
- Difficult to interpret model ...



Taxonomy of Machine Learning

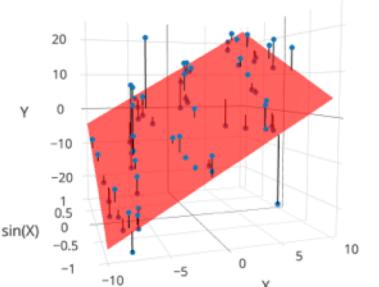


Labeled Data

Supervised Learning

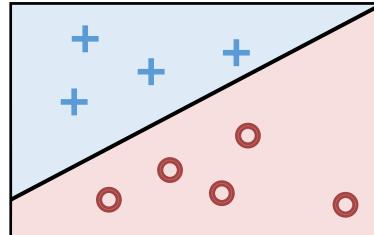
Quantitative Response

Regression



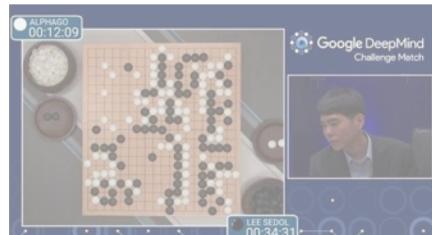
Categorical Response

Classification



Unlabeled Data

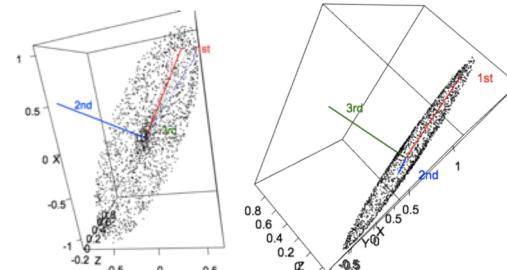
Reinforcement Learning (not covered)



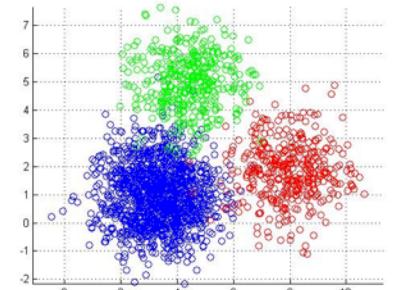
Alpha Go

Unsupervised Learning

Dimensionality Reduction



Clustering



Kinds of Classification

- **Binary** classification: Two classes
 - Examples: Spam/Not Spam, churn/stay
- **Multiclass** classification: Many classes (>2)
 - Examples: *Image labeling (Cat, Dog, Car)*, Next word in a sentence ...
- **Structured prediction** tasks (Classification)
 - Multiple related predictions
 - Examples: *Translation, Voice recognition*

Defining the Model

Logistic Regression

- Widely used models for **binary classification**:

x = “Get a FREE sample ...”

$$\phi(x) = [2.0, 0.0, \dots, 1.0, 0.5]$$

$$\rightarrow y = 1$$

1 = "Spam"
0 = "Ham"

- Models the probability of $y=1$ given x

Why is ham good
and spam bad? ...

(<https://www.youtube.com/watch?v=anwy2MPT5RE>)

$$\hat{P}_\theta(y = 1 | x) = \sigma(\phi(x)^T \theta) = \frac{1}{1 + \exp(-\phi(x)^T \theta)}$$

- Widely used models for **binary classification**:

x = “Get a FREE sample ...”

$$\rightarrow y = 1$$

$$\phi(x) = [2.0, 0.0, \dots, 1.0, 0.5]$$

1 = "Spam"
0 = "Ham"

- Models the probability of $y=1$ given x

Why is ham good
and spam bad? ...

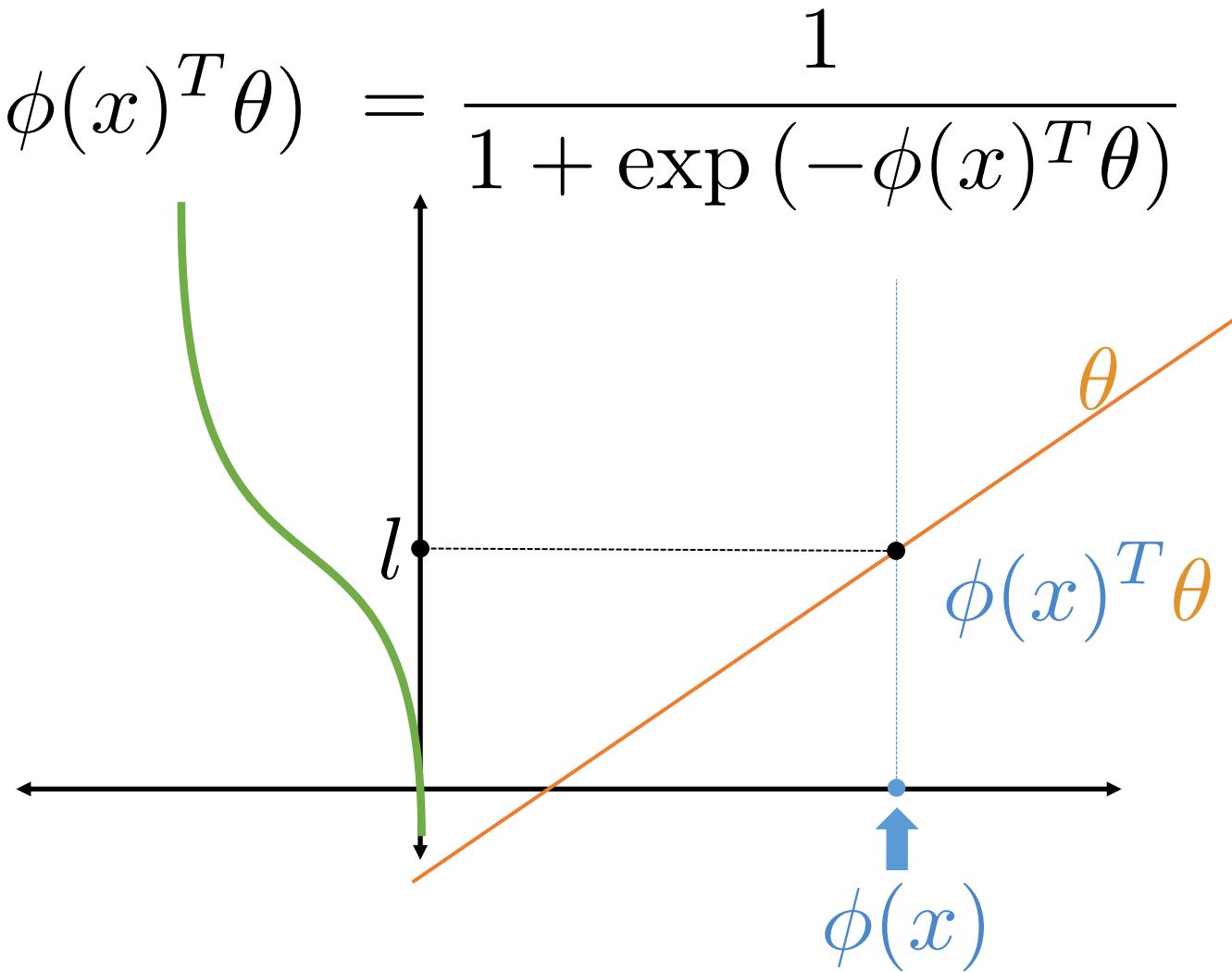
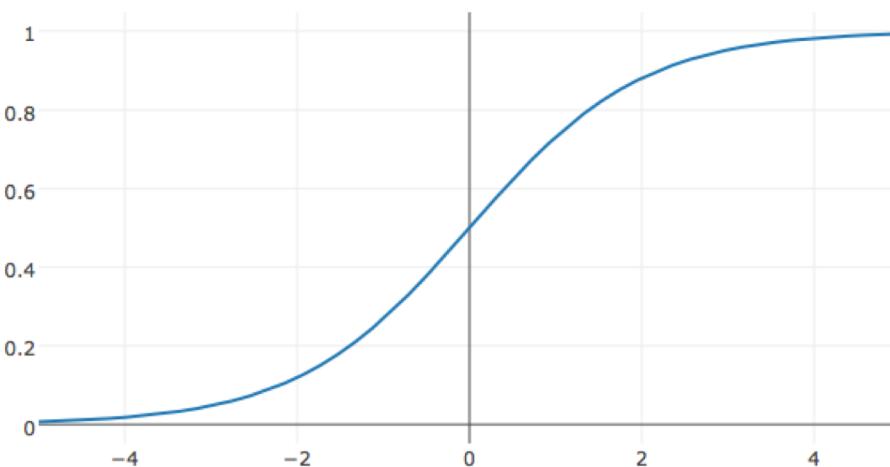
(<https://www.youtube.com/watch?v=anwy2MPT5RE>)

$$\hat{P}_\theta(y = 1 | x) = \sigma(\phi(x)^T \theta) = \frac{1}{1 + \exp(-\phi(x)^T \theta)}$$

$$\hat{P}_\theta(y = 0 | x) = 1 - \hat{P}_\theta(y = 1 | x)$$

Logistic Regression

Model: $\hat{P}_\theta(y = 1 | x) = \sigma(\phi(x)^T \theta) = \frac{1}{1 + \exp(-\phi(x)^T \theta)}$



Python Demo

The Logistic Regression Model

Model: $\hat{P}_\theta(y = 1 | x) = \sigma(\phi(x)^T \theta) = \frac{1}{1 + \exp(-\phi(x)^T \theta)}$

How do we fit the model to the data?

Defining the Loss

The Logistic Regression Model

Model: $\hat{P}_\theta(y = 1 | x) = \sigma(\phi(x)^T \theta) = \frac{1}{1 + \exp(-\phi(x)^T \theta)}$

The data:

x = “Get a FREE sample ...”

$$\phi(x) = [2.0, 0.0, \dots, 1.0, 0.5] \quad \rightarrow \quad y = 1$$

Corresponding probability:

$$P(y = 1 | x) = 1$$

$$P(y = 0 | x) = 0$$

Loss Function

- We want our model to be close to the data:

$$\hat{\mathbf{P}}_{\theta}(y = 1 | x) \approx \mathbf{P}(y = 1 | x)$$

- Kullback–Leibler (KL) Divergence

- For a **single** (x,y) data point

\Rightarrow Binary Classification

$$D_{KL}(\mathbf{P} || \hat{\mathbf{P}}_{\theta}) = \sum_{k=1}^{K=2} \mathbf{P}(y = k | x) \log \left(\frac{\mathbf{P}(y = k | x)}{\hat{\mathbf{P}}_{\theta}(y = k | x)} \right)$$

- Average KL Divergence for all the data:

- Kullback–Leibler (KL) Divergence

- For a **single** (x, y) data point

$$\mathbf{D}_{KL}(\mathbf{P} \parallel \hat{\mathbf{P}}_{\theta}) = \sum_{k=1}^{K=2} \mathbf{P}(y = k \mid x) \log \left(\frac{\mathbf{P}(y = k \mid x)}{\hat{\mathbf{P}}_{\theta}(y = k \mid x)} \right)$$

Binary Classification

- Average KL Divergence for all the data:

$$\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K \mathbf{P}(y_i = k \mid x_i) \log \left(\frac{\mathbf{P}(y_i = k \mid x_i)}{\hat{\mathbf{P}}_{\theta}(y_i = k \mid x_i)} \right)$$

$\log(a/b) = \log(a) - \log(b)$

Doesn't depend on θ ~~$\mathbf{P}(y_i = k \mid x_i) \log(\mathbf{P}(y_i = k \mid x_i))$~~

$$- \mathbf{P}(y_i = k \mid x_i) \log(\hat{\mathbf{P}}_{\theta}(y_i = k \mid x_i))$$

➤ Average cross entropy loss

$$\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^{K=2} \text{Binary Classification} - \mathbf{P}(y_i = k | x_i) \log(\hat{\mathbf{P}}_{\theta}(y_i = k | x_i))$$

$$\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n -\mathbf{P}(y_i = 1 | x_i) \log(\hat{\mathbf{P}}_{\theta}(y_i = 1 | x_i))$$

$$-\mathbf{P}(y_i = 0 | x_i) \log(\hat{\mathbf{P}}_{\theta}(y_i = 0 | x_i))$$

$\mathbf{P}(y_i = 1 x_i) = y_i$	$\hat{\mathbf{P}}_{\theta}(y_i = 1 x_i) = \sigma(\phi(x_i)^T \theta)$
$\mathbf{P}(y_i = 0 x_i) = (1 - y_i)$	$\hat{\mathbf{P}}_{\theta}(y_i = 0 x_i) = 1 - \sigma(\phi(x_i)^T \theta)$

➤ Average cross entropy loss

$K=2$ Binary Classification

$$\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^{K=2} - \mathbf{P}(y_i = k | x_i) \log \left(\hat{\mathbf{P}}_{\theta}(y_i = k | x_i) \right)$$
$$\begin{aligned} \arg \min_{\theta} & \frac{1}{n} \sum_{i=1}^n - \\ & y_i \quad \log \left(\sigma(\phi(x_i)^T \theta) \right) \\ & - (1 - y_i) \quad \log \left(1 - \sigma(\phi(x_i)^T \theta) \right) \end{aligned}$$

➤ Average cross entropy loss

$$\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^{K=2} \text{Binary Classification} - \mathbf{P}(y_i = k | x_i) \log (\hat{\mathbf{P}}_{\theta}(y_i = k | x_i))$$

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \log (\sigma(\phi(x_i)^T \theta)) +$$

Expanding

$$(1 - y_i) \log (1 - \sigma(\phi(x_i)^T \theta))$$

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \log (\sigma(\phi(x_i)^T \theta)) +$$

Grouping Terms

$$\log (1 - \sigma(\phi(x_i)^T \theta)) - y_i \log (1 - \sigma(\phi(x_i)^T \theta))$$

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \log (\sigma(\phi(x_i)^T \theta)) + \log (1 - \sigma(\phi(x_i)^T \theta)) - y_i \log (1 - \sigma(\phi(x_i)^T \theta))$$

Grouping Terms

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \log \left(\frac{\sigma(\phi(x_i)^T \theta)}{1 - \sigma(\phi(x_i)^T \theta)} \right) + \log (1 - \sigma(\phi(x_i)^T \theta))$$

$$\log \frac{\frac{1}{1+\exp(-\phi(x_i)^T \theta)} \times (1 + \exp (-\phi(x_i)^T \theta))}{1 - \frac{1}{1+\exp(-\phi(x_i)^T \theta)} \times (1 + \exp (-\phi(x_i)^T \theta))}$$

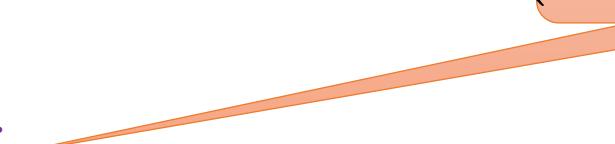
$\xrightarrow{\text{Alg.}}$

$$= \log \frac{1}{1 + \exp (-\phi(x_i)^T \theta) - 1} \xrightarrow{\text{Alg.}} = \log \exp (\phi(x_i)^T \theta) \xrightarrow{\text{Alg.}} = \phi(x_i)^T \theta$$

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \log \left(\frac{\sigma(\phi(x_i)^T \theta)}{1 - \sigma(\phi(x_i)^T \theta)} \right) + \log (1 - \sigma(\phi(x_i)^T \theta))$$

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \phi(x_i)^T \theta + \log (1 - \sigma(\phi(x_i)^T \theta))$$

$$1 - \frac{1}{1 + \exp(-\phi(x_i)^T \theta)}$$

Defn.
of σ

 $\times \exp(\phi(x_i)^T \theta)$

Alg.

$$= \frac{\exp(-\phi(x_i)^T \theta)}{1 + \exp(-\phi(x_i)^T \theta)}$$
Alg.

$$= \frac{1}{1 + \exp(\phi(x_i)^T \theta)}$$

Defn.
of σ
 $\times \exp(\phi(x_i)^T \theta)$

Definition

$$\sigma(\phi(x)^T \theta) = \frac{1}{1 + \exp(-\phi(x)^T \theta)}$$

$$= \sigma(-\phi(x_i)^T \theta)$$

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \phi(x_i)^T \theta + \log (\sigma(-\phi(x_i)^T \theta))$$

The Loss for Logistic Regression

- Average **cross entropy** (simplified):

$$\arg \min_{\theta} - \frac{1}{n} \sum_{i=1}^n y_i \phi(x_i)^T \theta + \log(\sigma(-\phi(x_i)^T \theta))$$

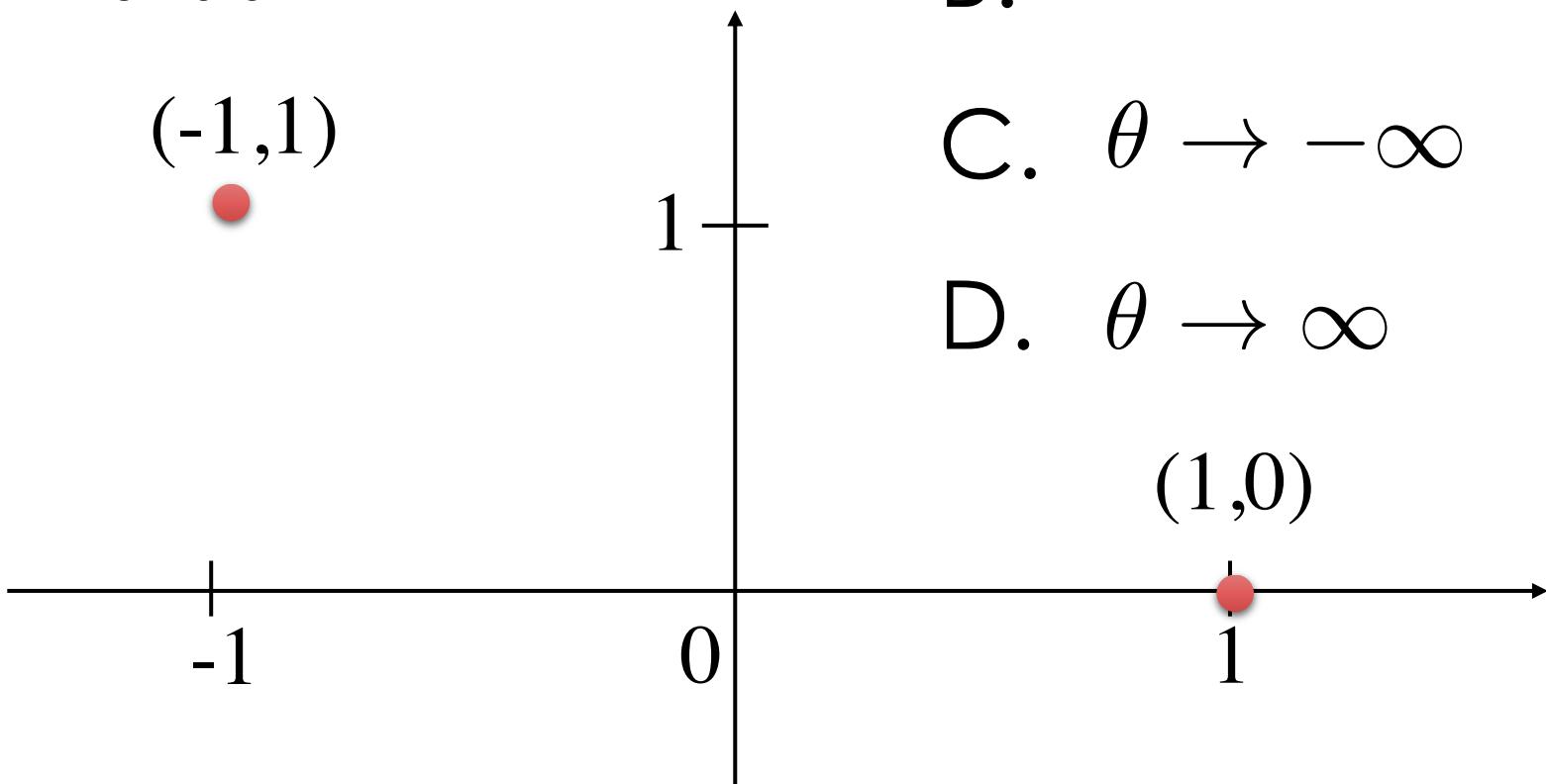
- Equivalent to (derived from) **minimizing the KL divergence**
- Also equivalent to **maximizing the log-likelihood of the data ...**
(not covered in DS100 this semester)

Is this loss function reasonable?

What is the value of θ ?

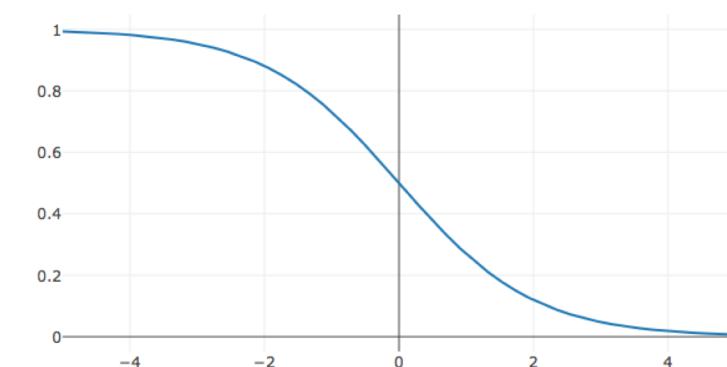
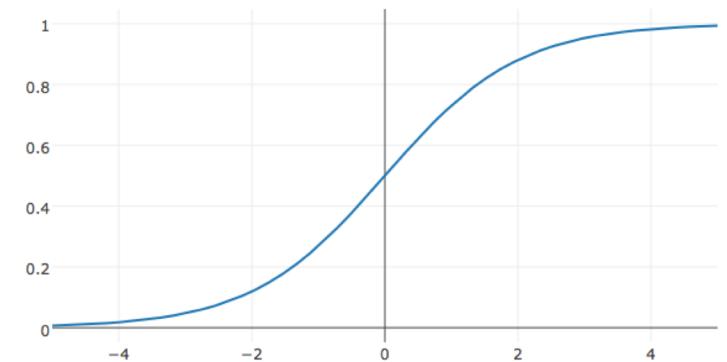
Assume: $\phi(x) = x$

The Data



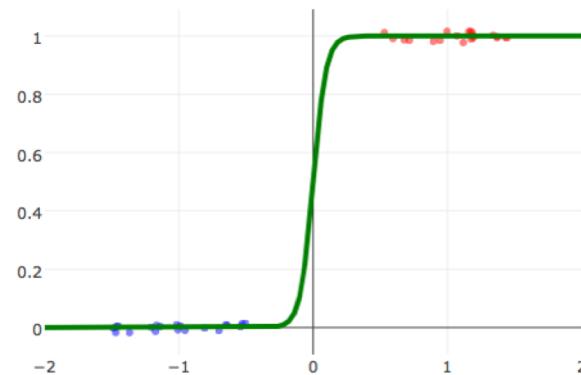
- A. $\theta = -1$
- B. $\theta = 1$
- C. $\theta \rightarrow -\infty$
- D. $\theta \rightarrow \infty$

$$\arg \min_{\theta} -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i) \cdot \theta + \log(\sigma(-\phi(x_i) \cdot \theta))$$



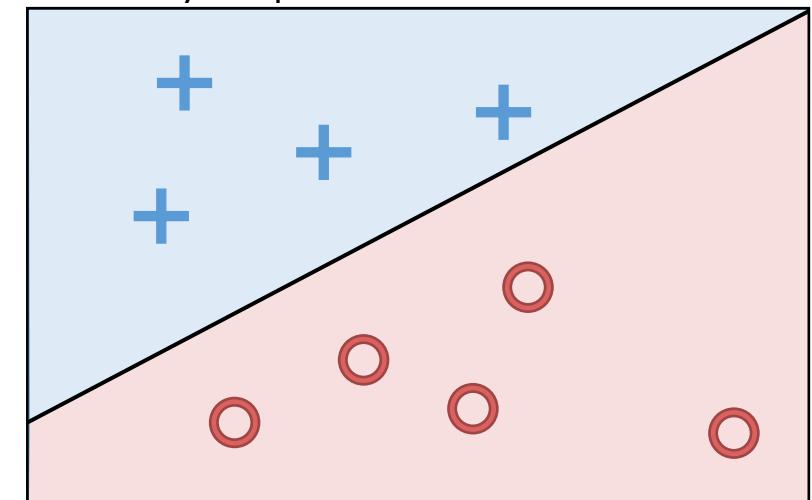
Linearly Separable Data

- A classification dataset is said to be linearly separable if there exists a hyperplane that separates the two classes.
- If data is linearly separable, logistic regression requires regularization

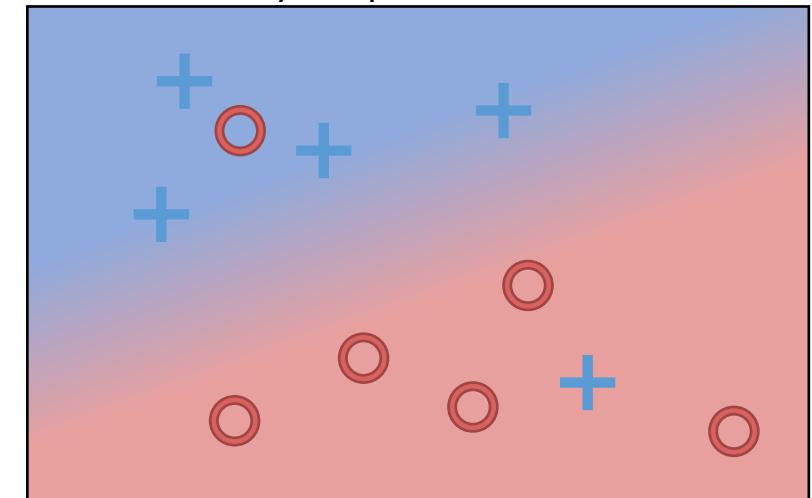


Weights go to infinity!

Linearly Separable Data



Not Linearly Separable Data



Logistic Loss Function

- Average KL divergence (simplified)

$$\arg \min_{\theta} -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i)^T \theta + \log (\sigma (-\phi(x_i)^T \theta))$$

- Take Derivative:

$$\begin{aligned}\nabla_{\theta} \mathbf{L}(\theta) &= -\frac{1}{n} \sum_{i=1}^n \nabla_{\theta} y_i \phi(x_i)^T \theta + \nabla_{\theta} \log (\sigma (-\phi(x_i)^T \theta)) \\ &= -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i) + \nabla_{\theta} \log (\sigma (-\phi(x_i)^T \theta))\end{aligned}$$

➤ Average KL divergence (simplified)

$$\arg \min_{\theta} -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i)^T \theta + \log (\sigma (-\phi(x_i)^T \theta))$$

➤ Take Derivative:

$$\begin{aligned}\nabla_{\theta} \mathbf{L}(\theta) &= -\frac{1}{n} \sum_{i=1}^n \nabla_{\theta} y_i \phi(x_i)^T \theta + \nabla_{\theta} \log (\sigma (-\phi(x_i)^T \theta)) \\ &= -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i) + \nabla_{\theta} \log (\sigma (-\phi(x_i)^T \theta)) \\ &= -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i) + \frac{1}{\sigma (-\phi(x_i)^T \theta)} \nabla_{\theta} \sigma (-\phi(x_i)^T \theta)\end{aligned}$$

➤ Take Derivative:

$$\nabla_{\theta} \mathbf{L}(\theta) = -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i) + \frac{1}{\sigma(-\phi(x_i)^T \theta)} \nabla_{\theta} \sigma(-\phi(x_i)^T \theta)$$

Useful Identity

$$\frac{\partial}{\partial t} \sigma(t) = \frac{\partial}{\partial t} \frac{1}{1 + e^{-t}} \stackrel{\text{Chain Rule}}{=} \frac{-1}{(1 + e^{-t})^2} \frac{\partial}{\partial t} (1 + e^{-t})$$

$$\stackrel{\text{Chain Rule}}{=} \frac{e^{-t}}{(1 + e^{-t})^2} \stackrel{\text{Alg.}}{=} \left(\frac{1}{1 + e^{-t}} \right) \left(\frac{e^{-t}}{1 + e^{-t}} \right)$$

$$\stackrel{\text{Alg.}}{=} \left(\frac{1}{1 + e^{-t}} \right) \left(\frac{1}{e^t + 1} \right) \stackrel{\text{Defn. of } \sigma}{=} \sigma(t) \sigma(-t)$$

➤ Take Derivative:

$$\nabla_{\theta} \mathbf{L}(\theta) = -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i) + \frac{1}{\sigma(-\phi(x_i)^T \theta)} \nabla_{\theta} \sigma(-\phi(x_i)^T \theta)$$

Useful Identity

$$\frac{\partial}{\partial t} \sigma(t) = \sigma(t)\sigma(-t)$$

$$= -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i) + \frac{\sigma(-\phi(x_i)^T \theta)}{\sigma(-\phi(x_i)^T \theta)} \sigma(\phi(x_i)^T \theta) \nabla_{\theta} (-\phi(x_i)^T \theta)$$

$$= -\frac{1}{n} \sum_{i=1}^n (y_i - \sigma(\phi(x_i)^T \theta)) \phi(x_i)$$

Logistic Loss Function

- Average KL divergence (simplified)

$$\arg \min_{\theta} -\frac{1}{n} \sum_{i=1}^n y_i \phi(x_i)^T \theta + \log (\sigma (-\phi(x_i)^T \theta))$$

- Take Derivative:

$$\nabla_{\theta} \mathbf{L}(\theta) = -\frac{1}{n} \sum_{i=1}^n (y_i - \sigma (\phi(x_i)^T \theta)) \phi(x_i)$$

- Set derivative = 0 and solve for θ
 - No general analytic solution
 - Solved using numeric methods

The Gradient Descent Algorithm

$$\theta^{(0)} \leftarrow \text{initial vector (random, zeros ...)}$$

For τ from 0 to convergence:

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau) \left(\nabla_{\theta} \mathbf{L}(\theta) \middle| \begin{array}{l} \text{Evaluated} \\ \text{at} \\ \theta = \theta^{(\tau)} \end{array} \right)$$

- $\rho(\tau)$ is the step size (learning rate)
 - typically $1/\tau$
- Converges when gradient is ≈ 0 (or we run out of patience)

Gradient Descent for Logistic Regression

Logistic Regression

$$\theta^{(0)} \leftarrow \text{initial vector (random, zeros ...)}$$

For τ from 0 to convergence:

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau) \left(\frac{1}{n} \sum_{i=1}^n \left(\sigma\left(\phi(x_i)^T \theta^{(\tau)}\right) - y_i \right) \phi(x_i) \right)$$

- $\rho(\tau)$ is the step size (learning rate)
 - typically $1/\tau$
- Converges when gradient is ≈ 0 (or we run out of patience)

Demo!

Stochastic Gradient Descent

- For many learning problems the gradient is a sum:

$$\nabla_{\theta} \mathbf{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (\sigma(\phi(x_i)^T \theta) - y_i) \phi(x_i)$$

- For large n this can be costly
- What if we approximated the gradient by looking at a few random points:

$$\nabla_{\theta} \mathbf{L}(\theta) \approx \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\sigma(\phi(x_i)^T \theta) - y_i) \phi(x_i)$$

- What if we approximated the gradient by looking at a few random points:

$$\nabla_{\theta} \mathbf{L}(\theta) \approx \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\sigma(\phi(x_i)^T \theta) - y_i) \phi(x_i)$$

Batch
Size

Random sample
of records

- This is a reasonable estimator for the gradient
 - Unbiased ...
- Often batch size is one! (why is this helpful)
 - Fast to compute!
- A key ingredient in the recent success of deep learning

Stochastic Gradient Descent

$\theta^{(0)} \leftarrow$ initial vector (random, zeros ...)

For τ from 0 to convergence:

$\mathcal{B} \sim$ Random subset of indices

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau) \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\theta} \mathbf{L}_i(\theta) \Big|_{\theta=\theta^{(\tau)}} \right)$$

Decomposable
Loss

$$\mathbf{L}(\theta) = \sum_{i=1}^n \mathbf{L}_i(\theta) = \sum_{i=1}^n \mathbf{L}(\theta, x_i, y_i)$$

Loss can be written as a sum of the loss on each record.

$$\theta^{(0)} \leftarrow \text{initial vector (random, zeros ...)}$$

For τ from 0 to convergence:

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau) \left(\frac{1}{n} \sum_{i=1}^n \nabla_{\theta} \mathbf{L}_i(\theta) \Big|_{\theta=\theta^{(\tau)}} \right)$$

$$\theta^{(0)} \leftarrow \text{initial vector (random, zeros ...)}$$

For τ from 0 to convergence:

$\mathcal{B} \sim$ Random subset of indices

$$\theta^{(\tau+1)} \leftarrow \theta^{(\tau)} - \rho(\tau) \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\theta} \mathbf{L}_i(\theta) \Big|_{\theta=\theta^{(\tau)}} \right)$$

Very Similar
Algorithms

Stochastic Gradient Intuition



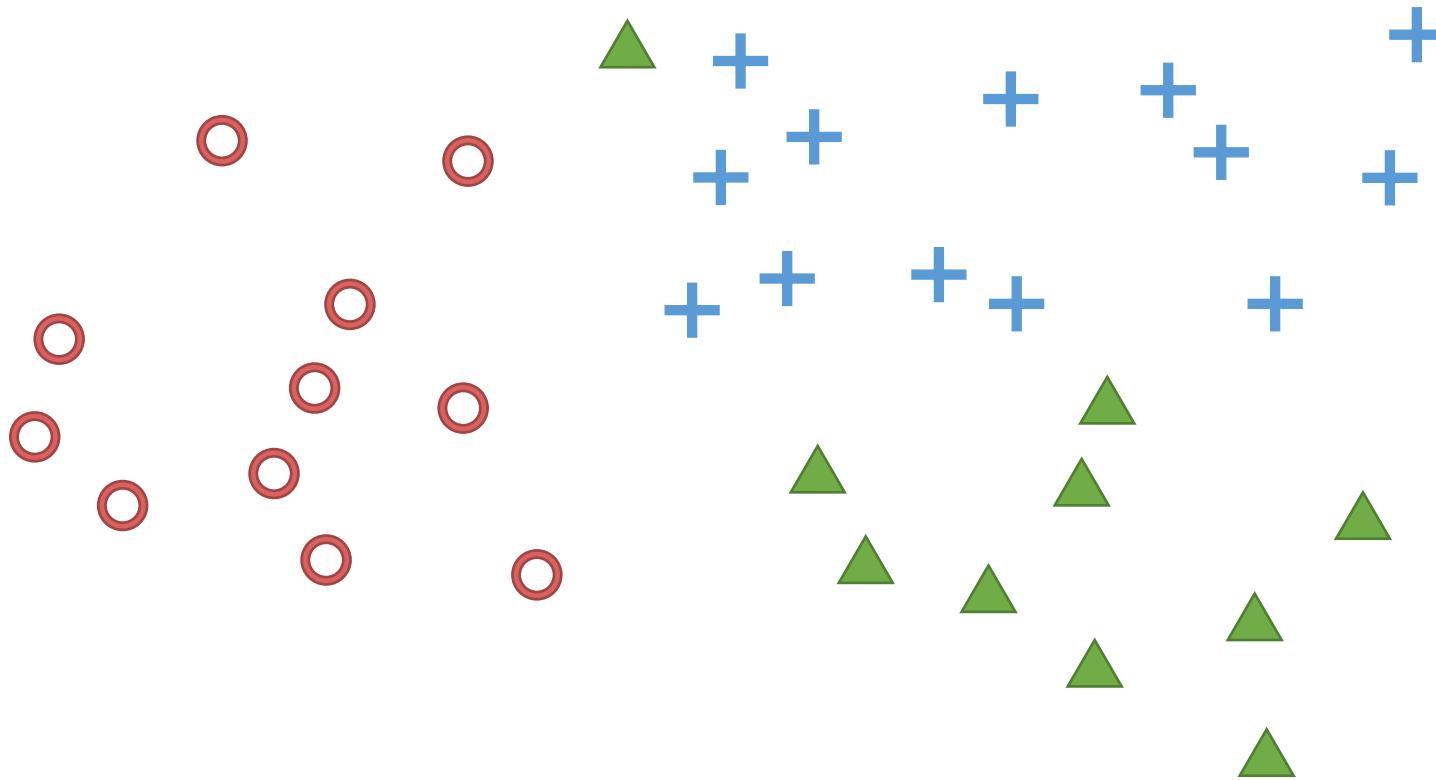
"If I keep going downhill on average then I will get to the bottom."

Why might this reasoning fail?

Python Demo!

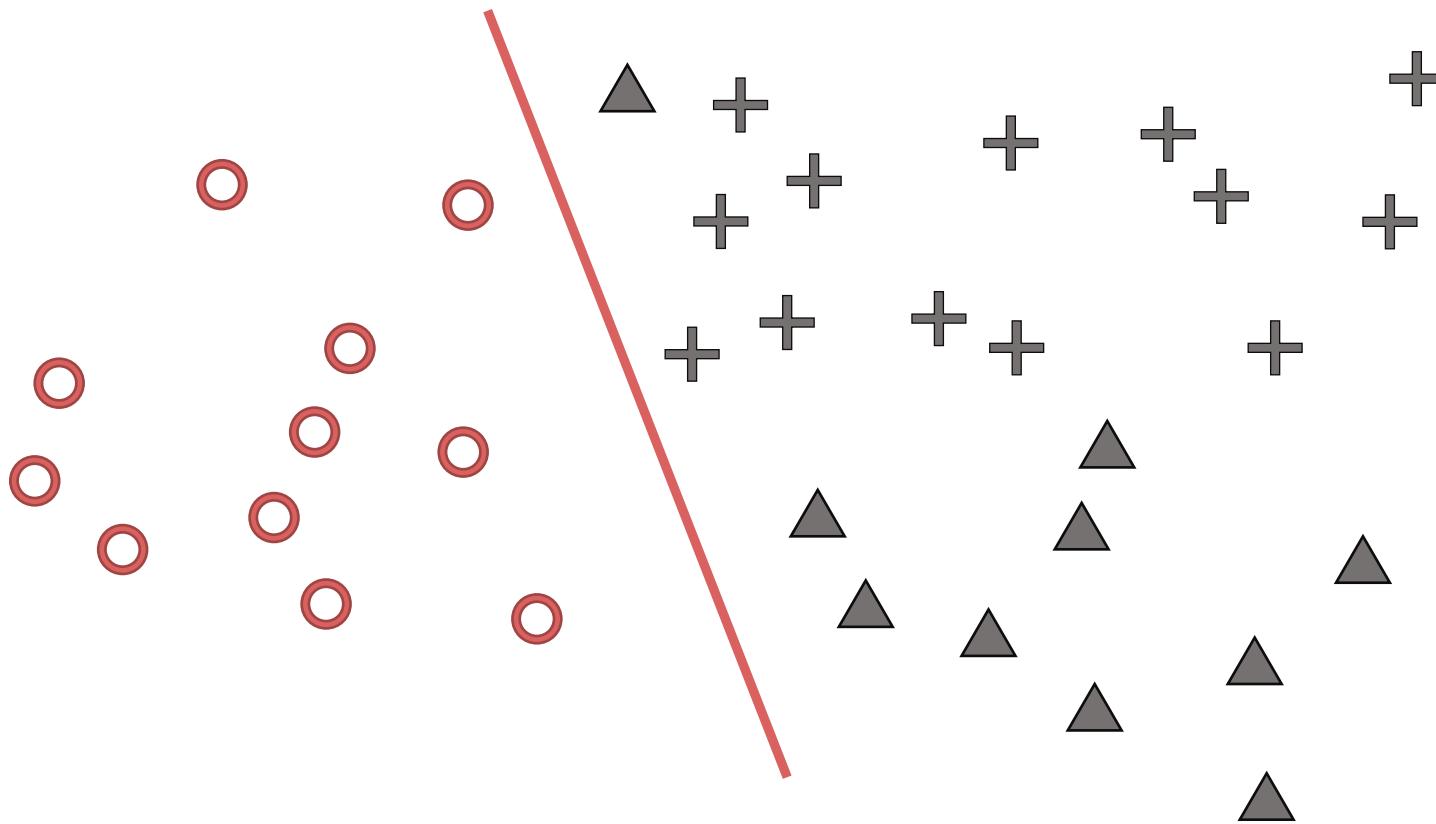
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



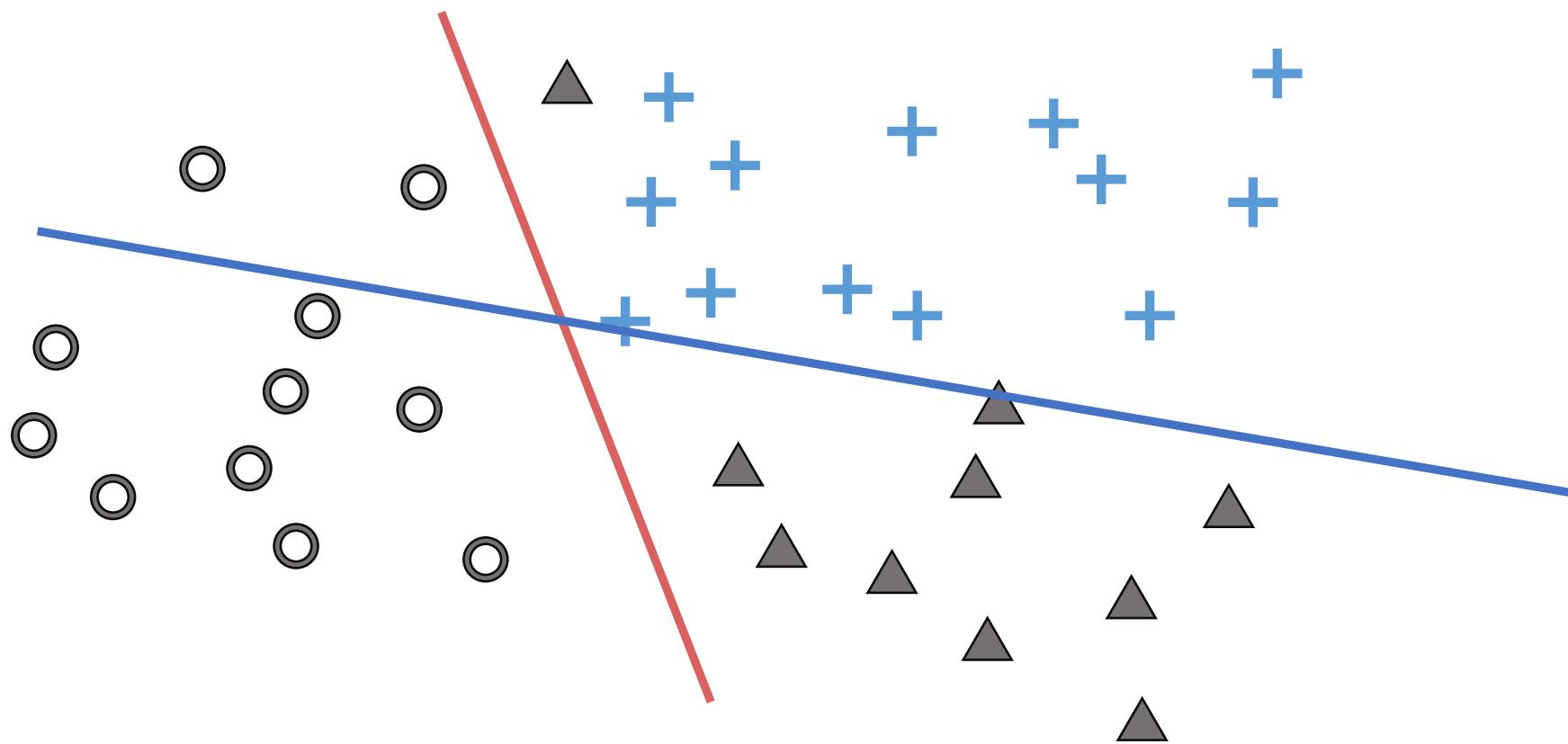
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



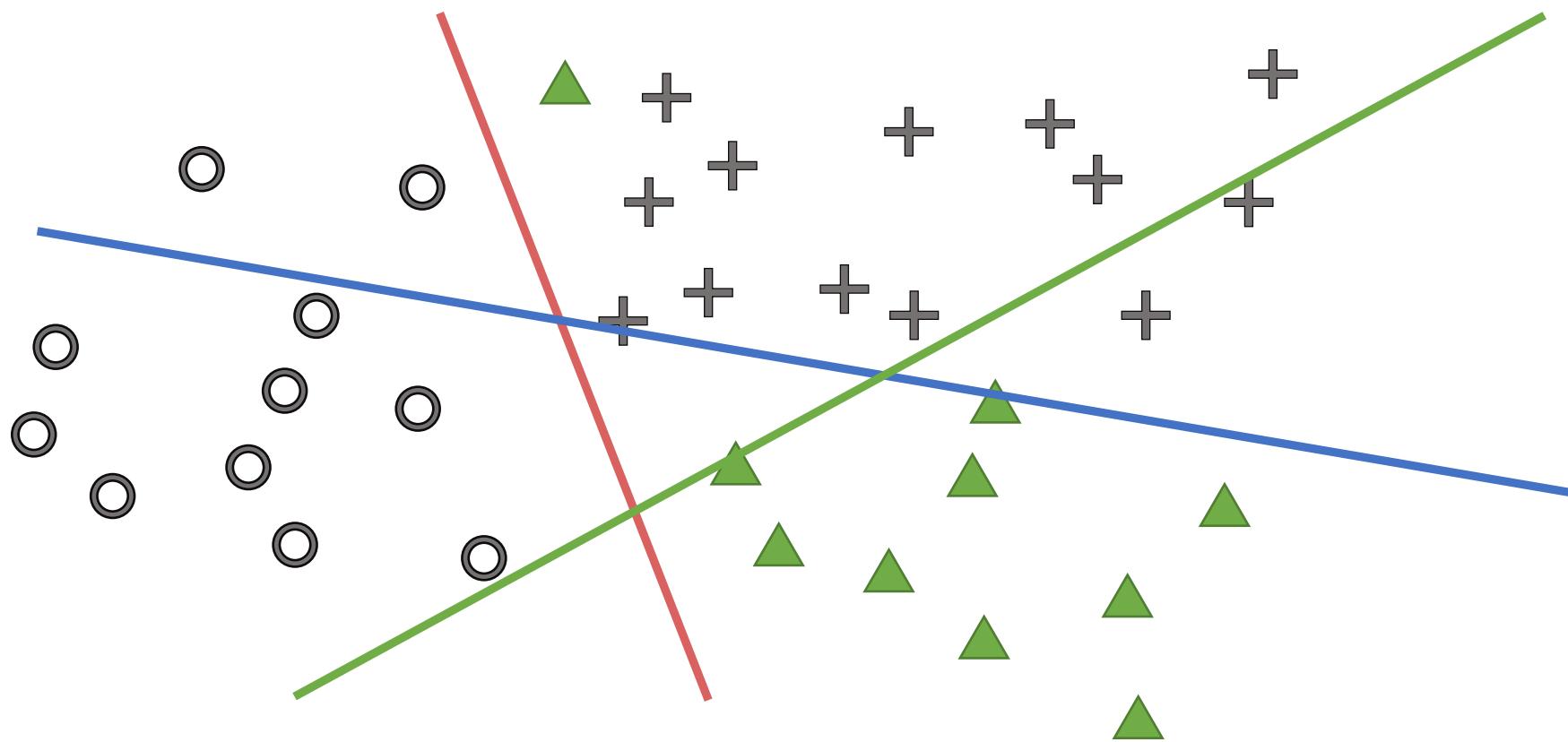
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



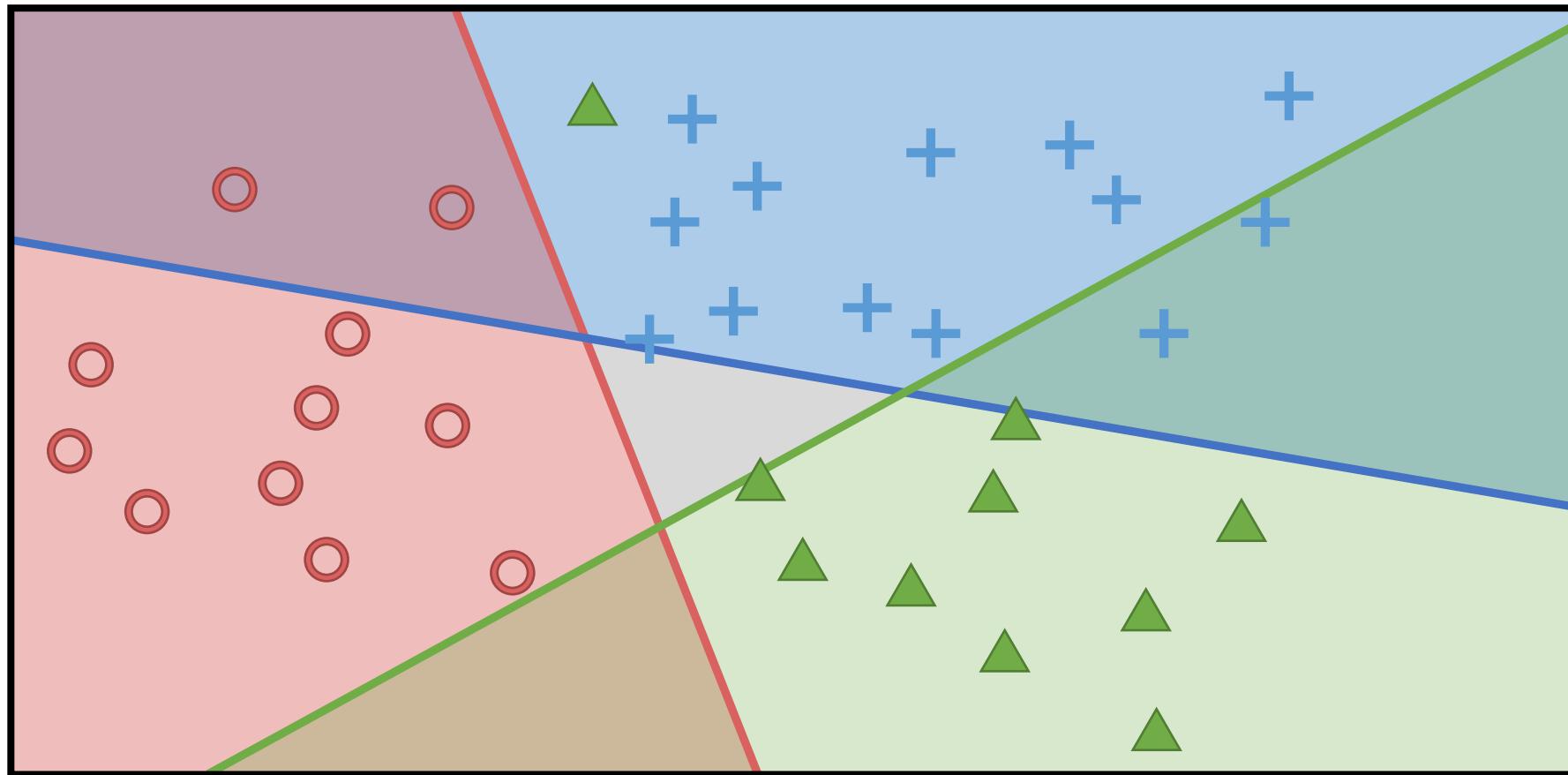
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



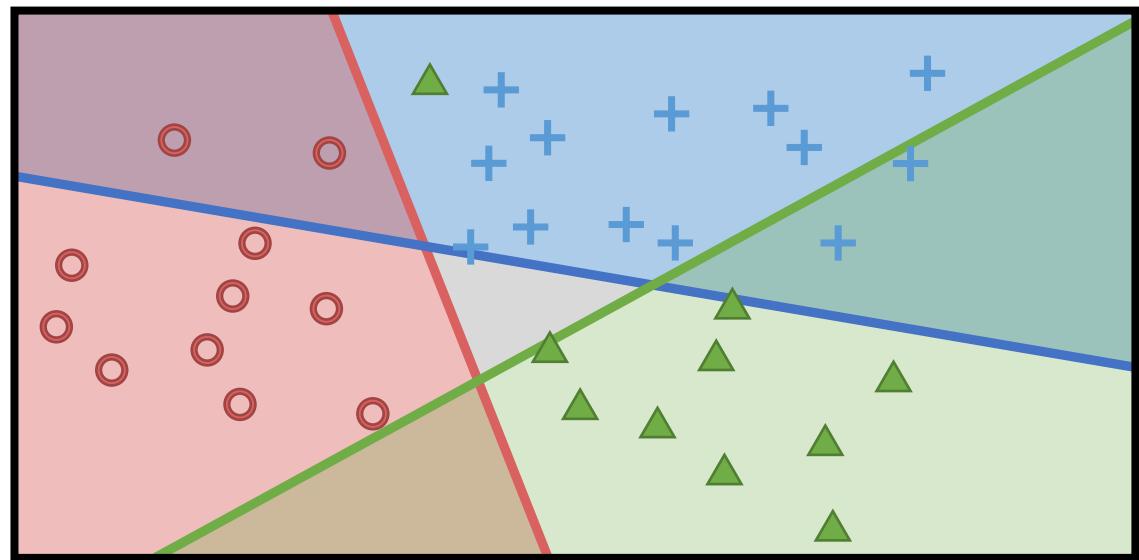
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class
 - Class with highest confidence wins
 - Need to address class imbalance issue
- **Soft-Max** multiclass classification



➤ **Soft-Max** multiclass classification

$$\mathbf{P}(Y = j \mid x) = \frac{\exp(x^T \theta^{(j)})}{\sum_{m=1}^k \exp(x^T \theta^{(m)})}$$

- Separate $\theta^{(j)} \in \mathbb{R}^p$ for each class
- Trained using gradient descent methods
- Over parameterized. Why?
 - k sets of parameters one for each class
 - Only need $k-1$ parameters

$$\mathbf{P}(y = k \mid x) = 1 - \sum_{j=1}^K \mathbf{P}(y = j \mid x)$$

- Often use k parameters + regularization to address lack of identifiability.

Python Demo!

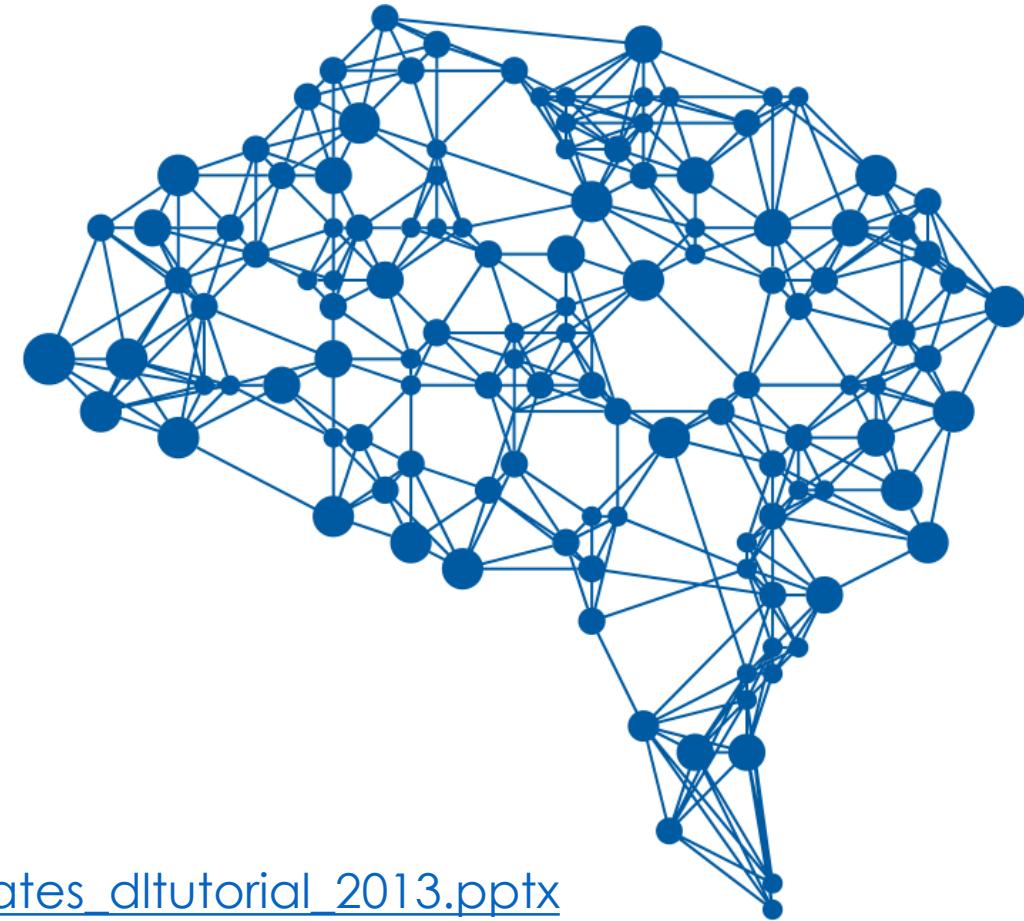
Deep Learning

Overview

Bonus Material

Borrowed heavily from excellent talks by:

- **Adam Coates:** http://ai.stanford.edu/~acoates/coates_dltutorial_2013.pptx
- **Fei-Fei Li and Andrej Karpathy:** <http://cs231n.stanford.edu/syllabus.html>

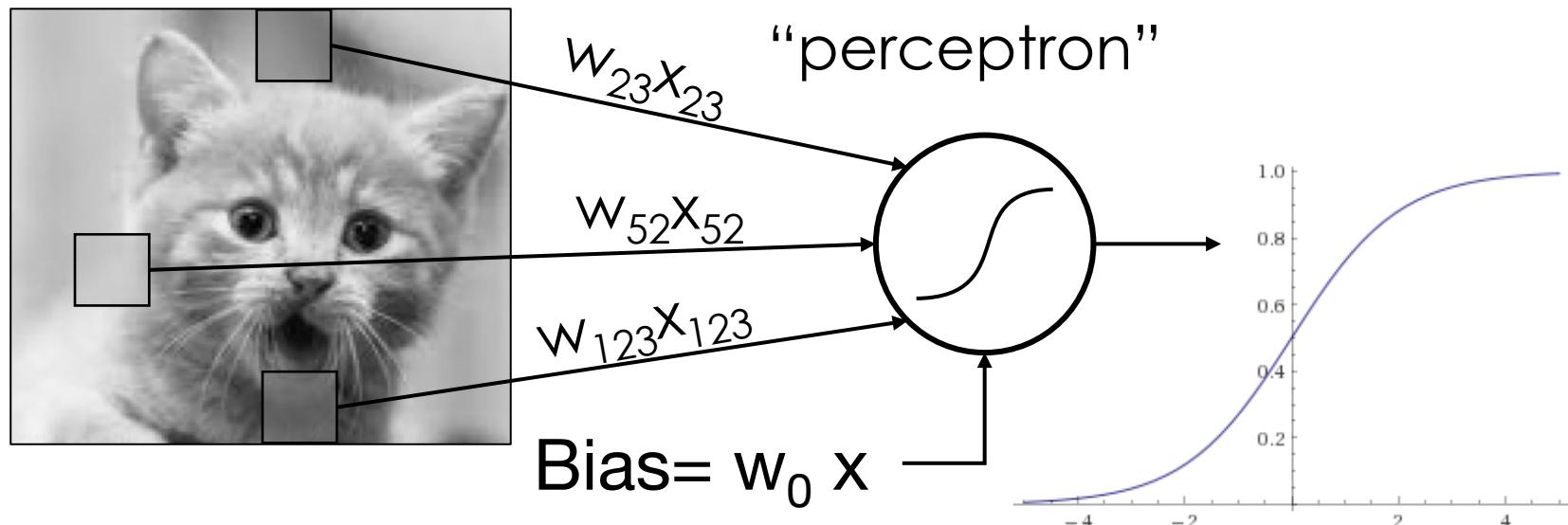


Logistic Regression as a “Neuron”

- Consider the simple function family:

$$\sigma(u) = \frac{1}{1 + \exp(-u)}$$

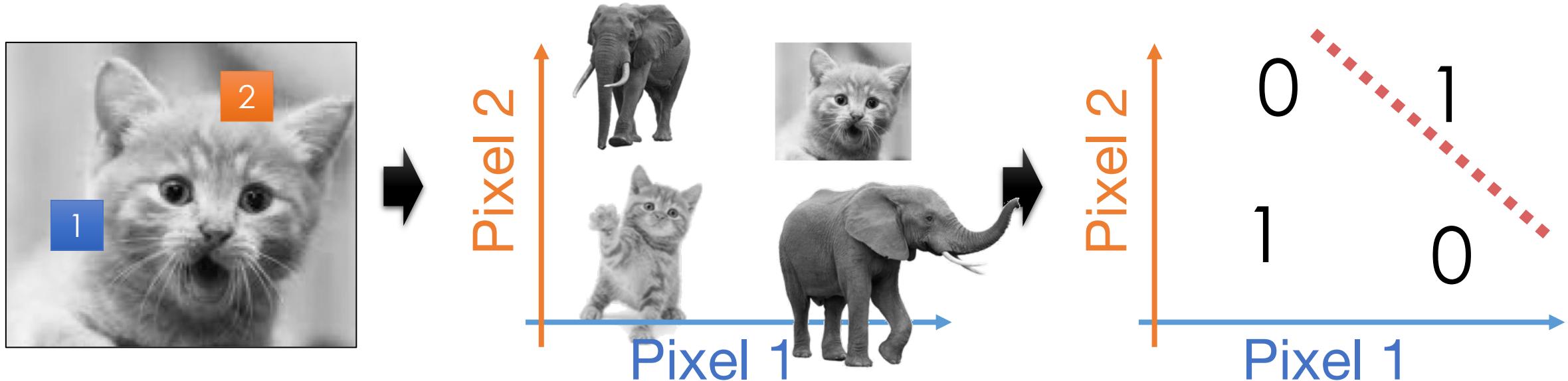
$$f_w(x) = \sigma(w^T x) = \sigma\left(\sum_{j=1}^d w_j x_j\right) = P(y = 1 | x)$$



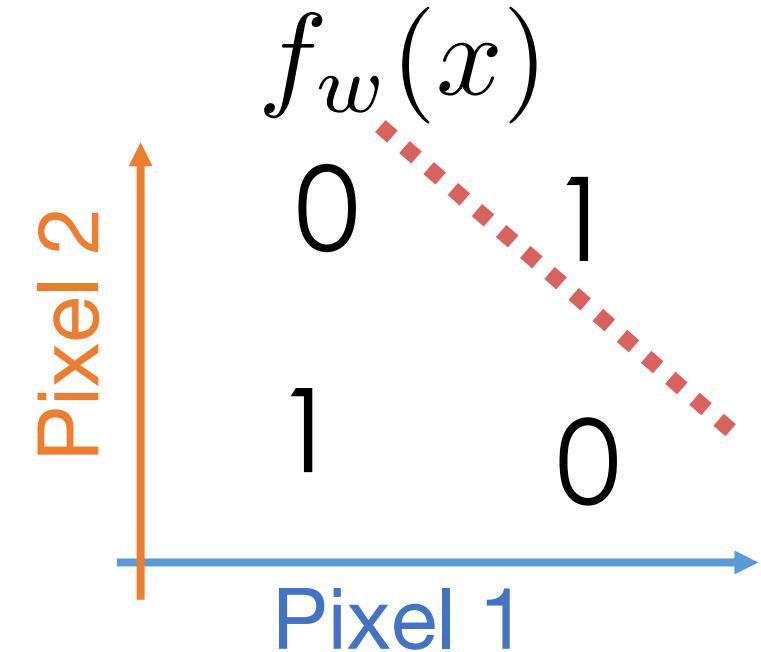
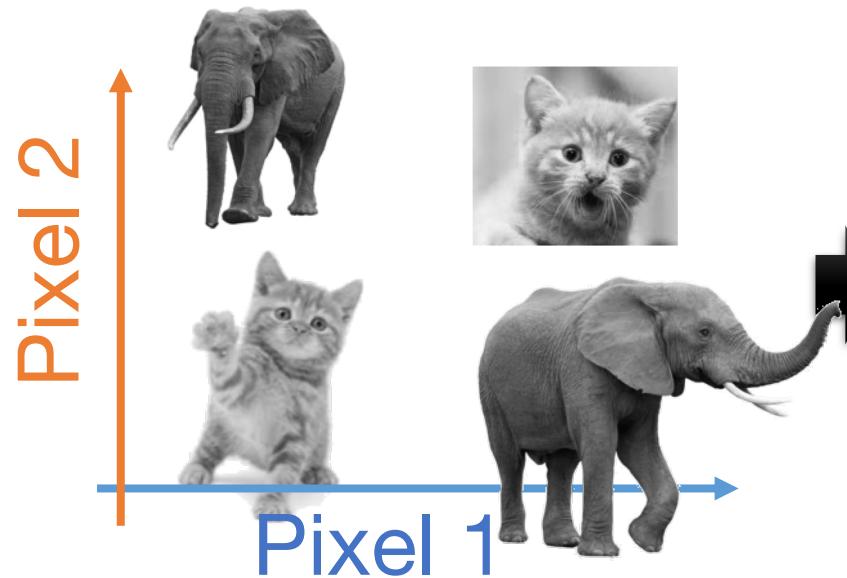
*Neuron “fires”
if weighted
sum of input is
greater than
zero.*

Logistic Regression: Strengths and Limitations

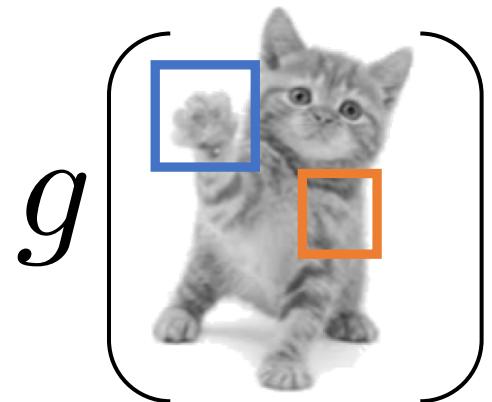
- Widely used machine learning technique
 - convex → efficient to learn
 - easy to interpret model weights
 - works well given good features
- Limitations:
 - Restricted to linear relationships → sensitive to choice of features



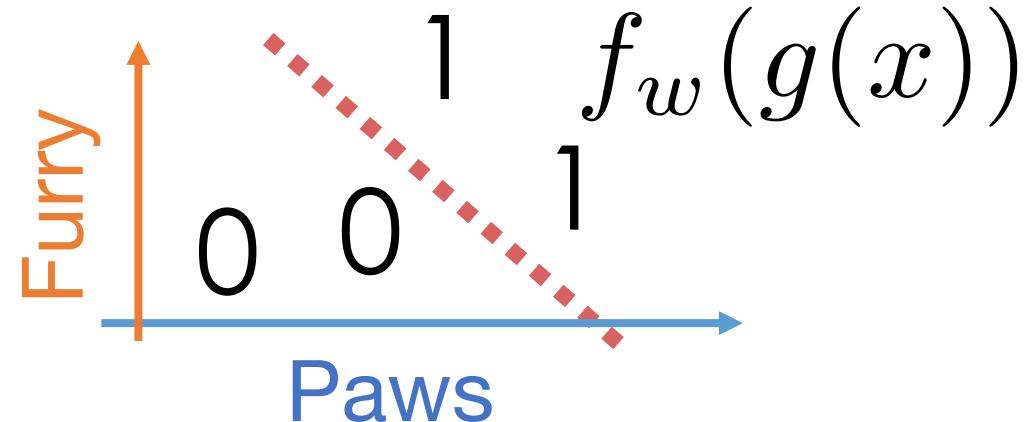
Feature Engineering



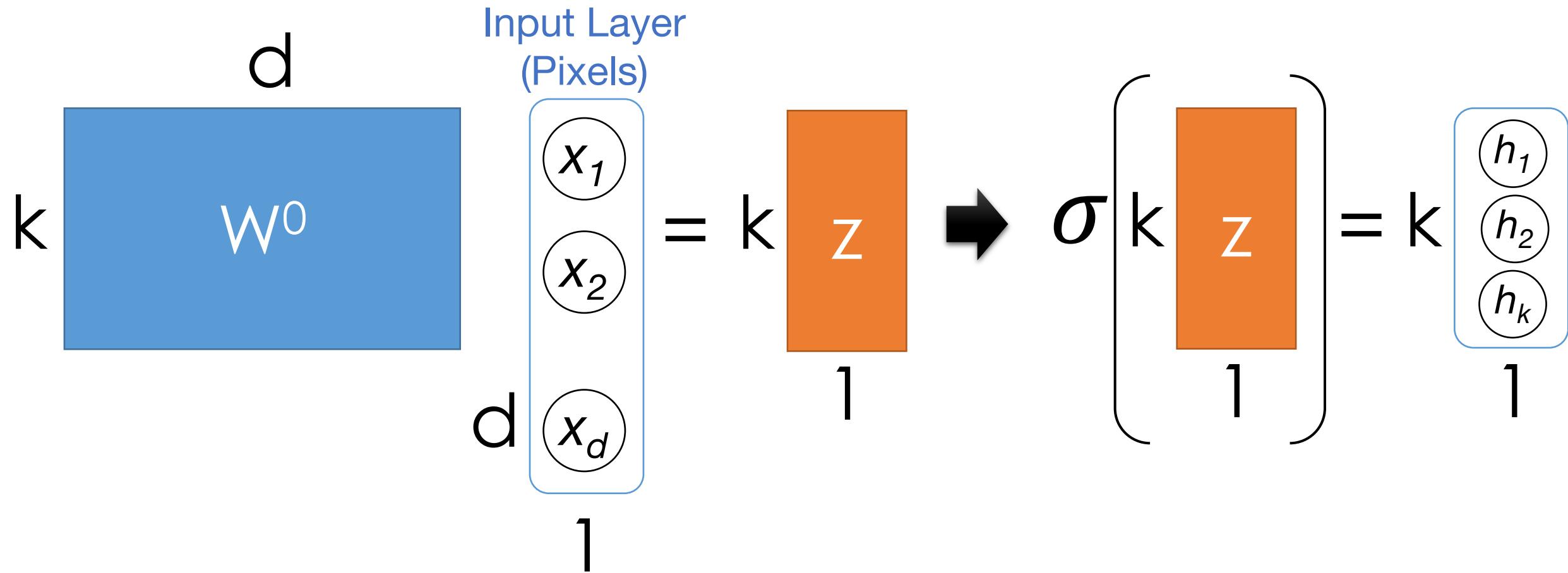
- Rather than use raw **pixels build/train feature functions**:



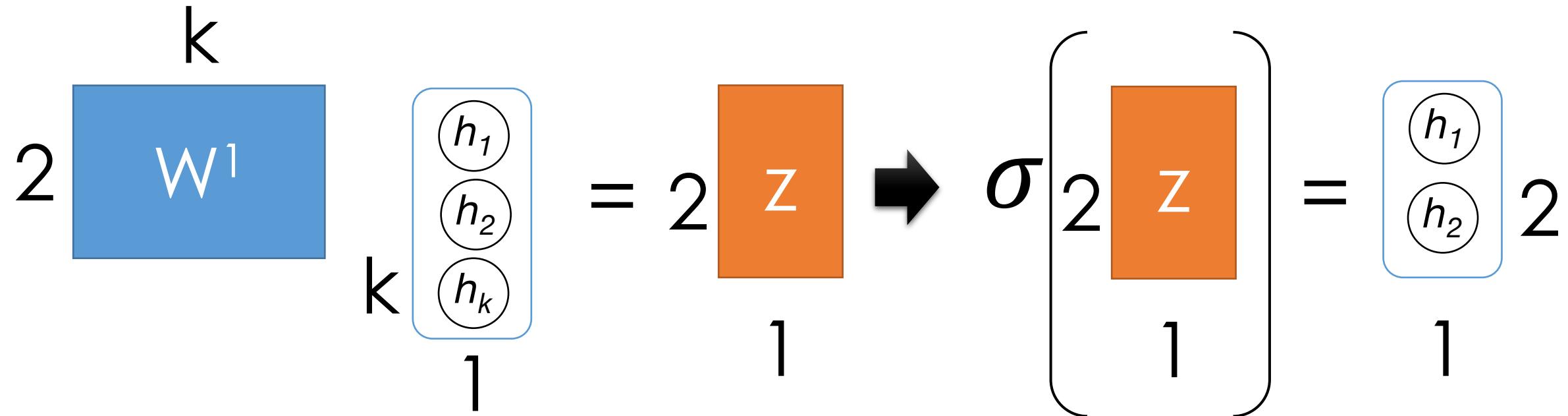
$$g = (\text{Paws}, \text{Furry})$$



Composition Linear Models and Nonlinearities



Composition Linear Models and Nonlinearities

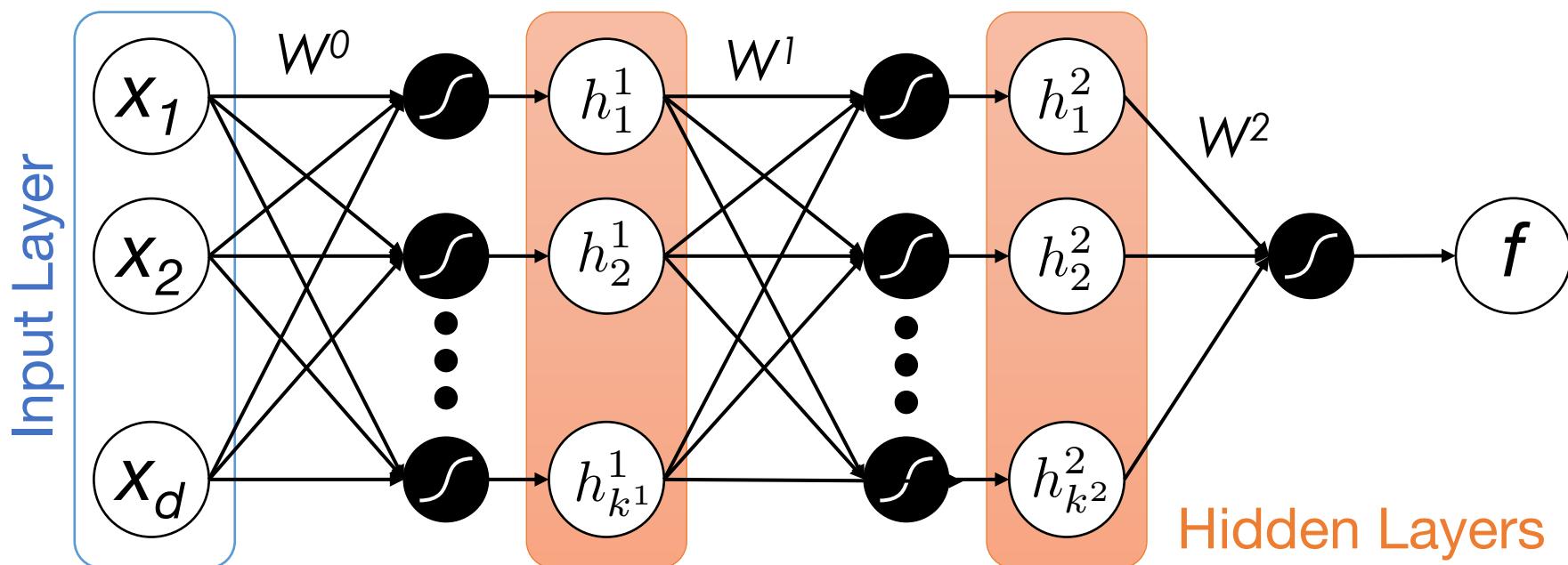


Neural Networks

- Composing “perceptrons”

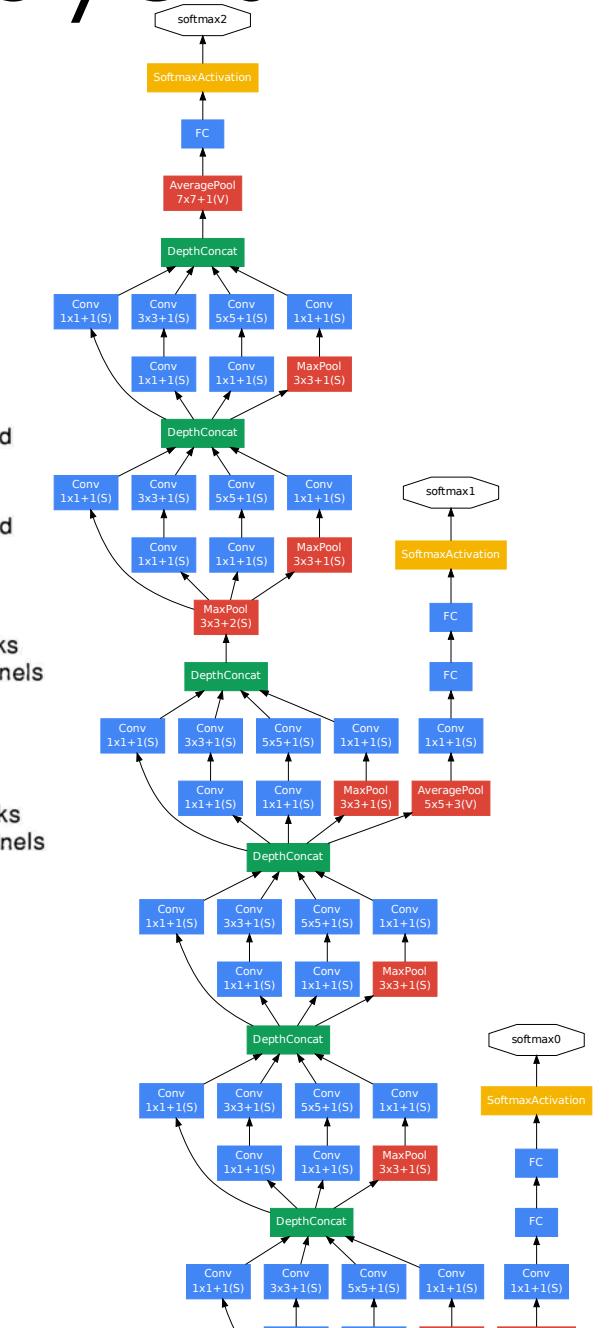
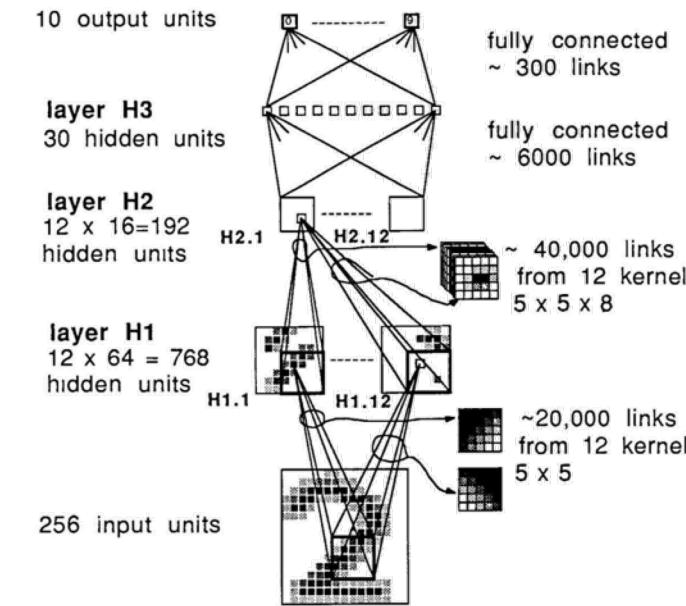
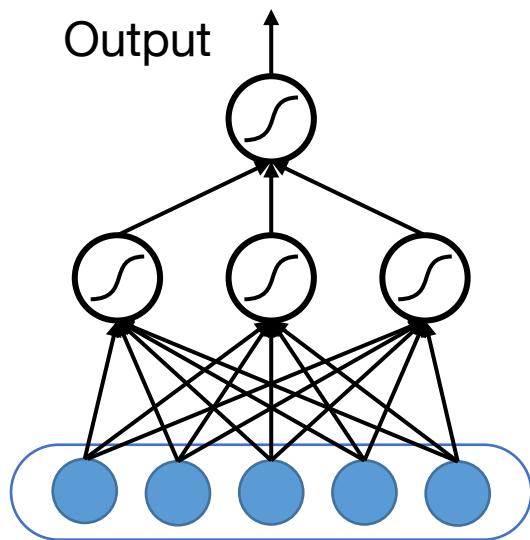
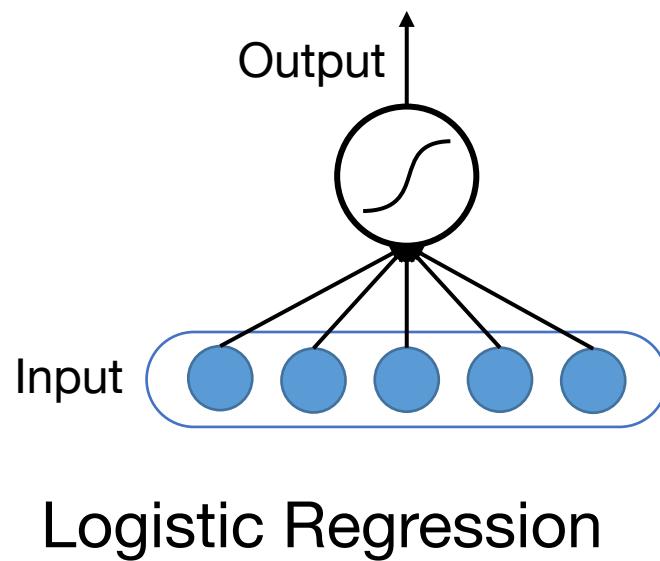
$$x \rightarrow \sigma(W^0 x) \rightarrow h^1 \rightarrow \sigma(W^1 h^1) \rightarrow h^2 \rightarrow \sigma(W^2 h^2) \rightarrow f$$

$$y = f_{W^0, W^1, W^2}(x) = \sigma(W^2 \sigma(W^1 \sigma(W^0 x)))$$

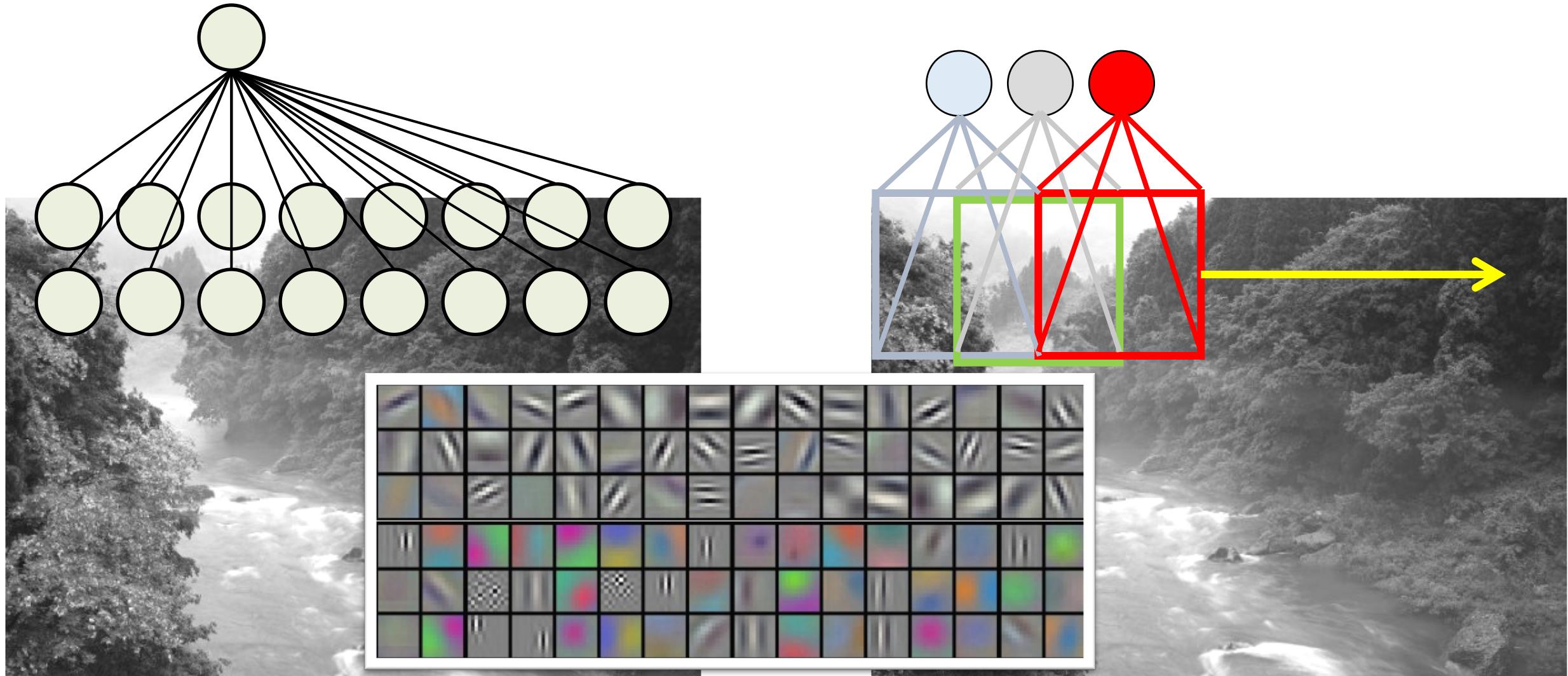


Deep Learning → Many hidden layers

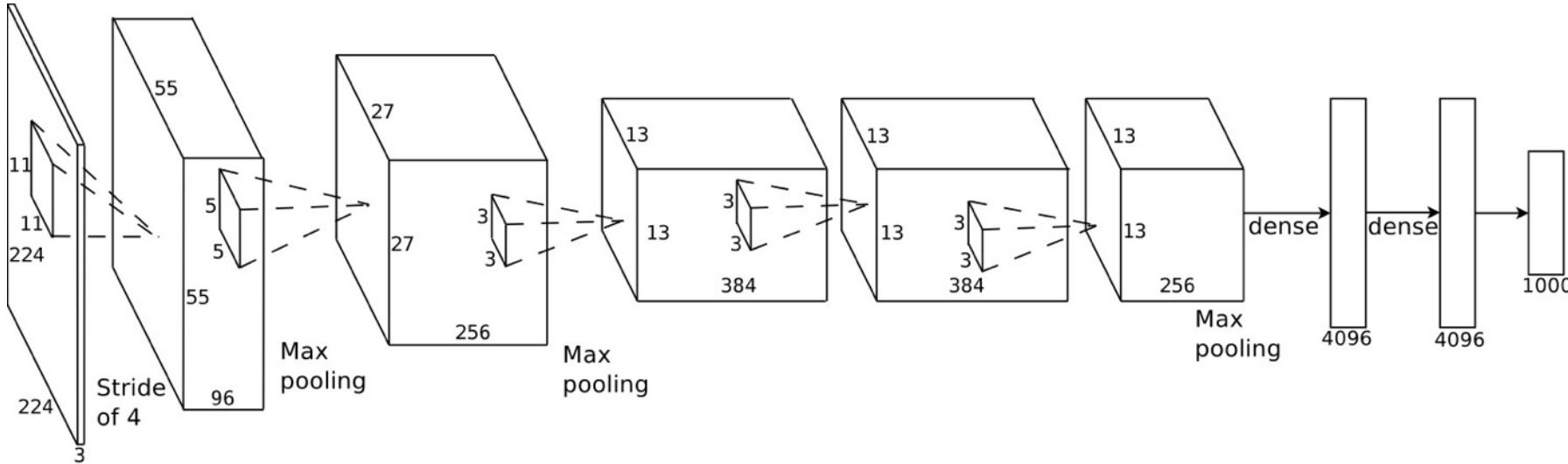
...



Convolutional Neural Networks: Exploiting Spatial Sparsity

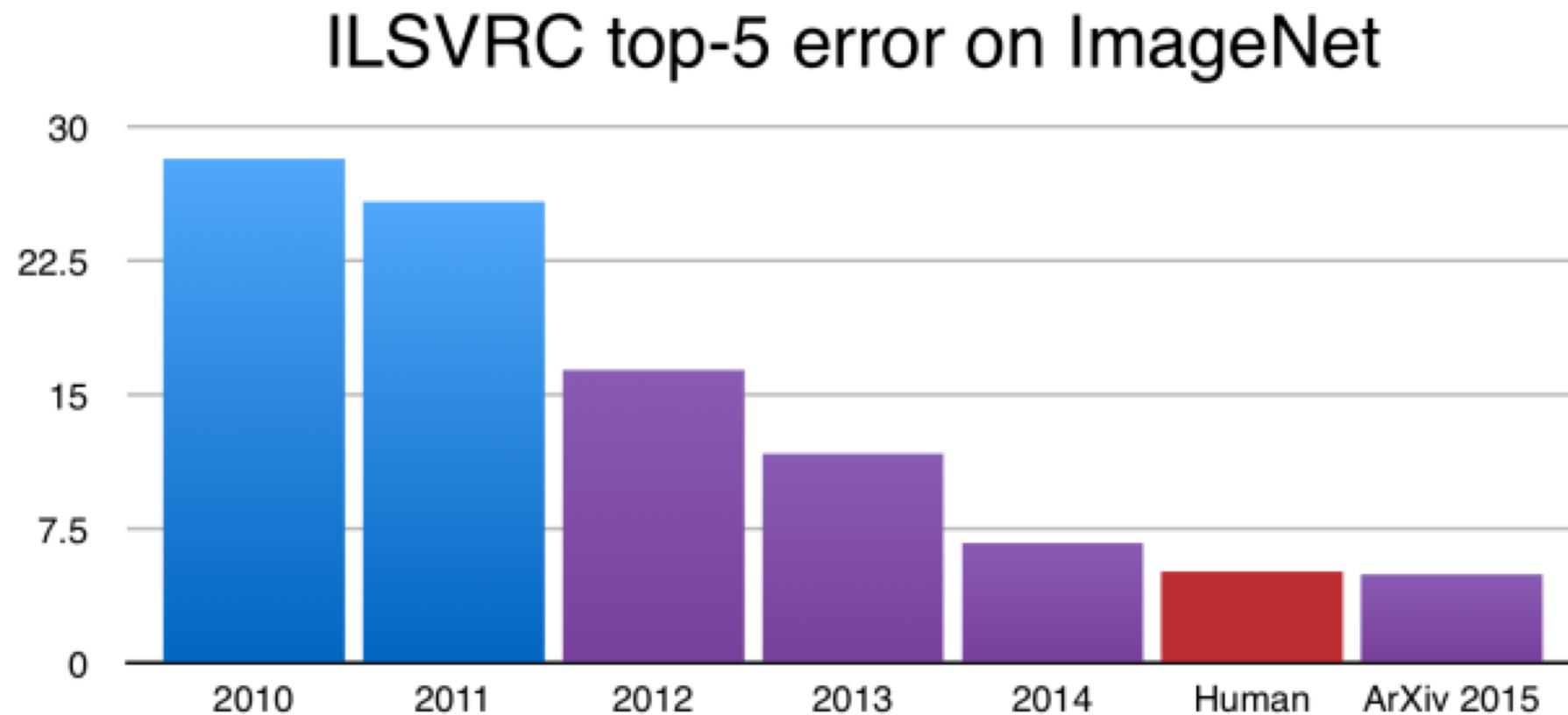


Example: AlexNet (Krizhevsky et al., NIPS 2012)

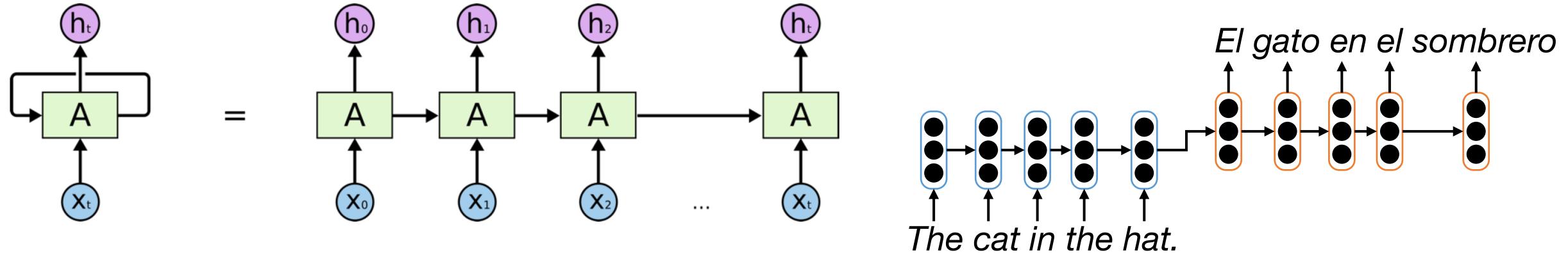


- Introduced in 2012, significantly outperformed state-of-the-art (top 5 error of 16% compared to runner-up with 26% error)
- Covered in reading ...

Improvement on ImageNet Benchmark



Recurrent Neural Networks: Modeling Sequence Structure

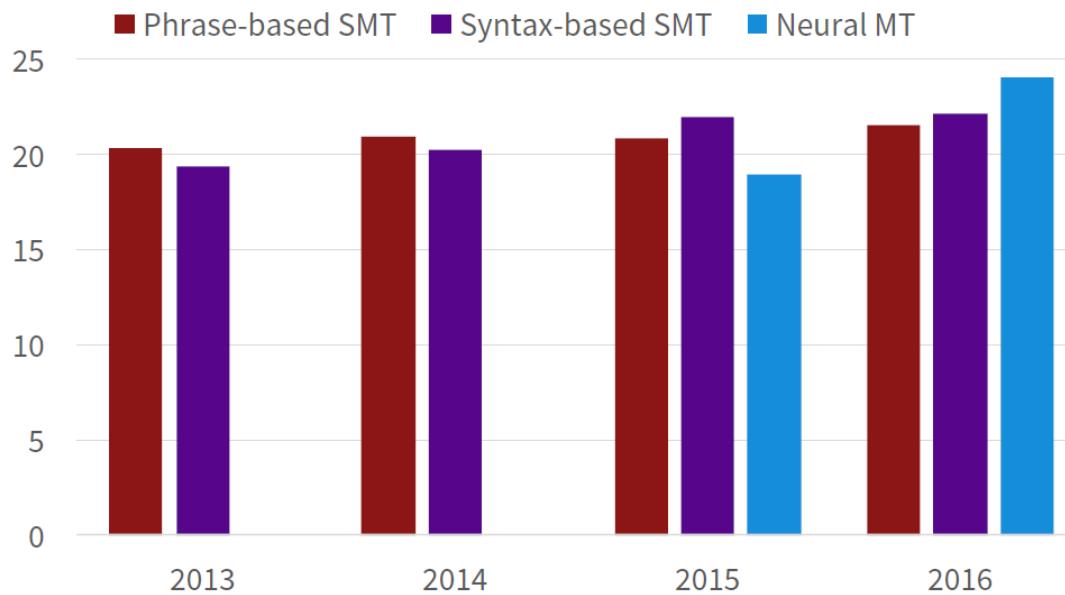


- State of the art in speech recognition and machine translation
 - Required LSTM and GRU to address long dependencies
- Similar to the HMM from classical Bayesian ML

Improvements in Machine Translation & Automatic Speech Recognition

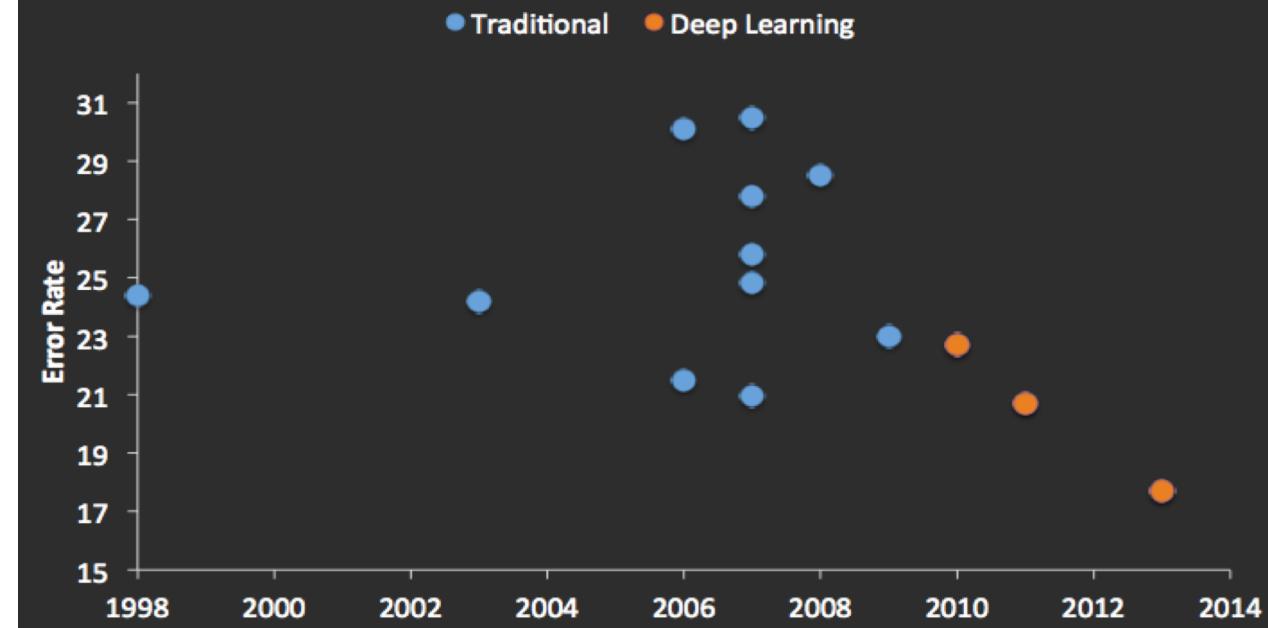
Progress in Machine Translation

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]



From [Sennrich 2016, http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf]

TIMIT Speech Recognition



Interested in Deep Learning?

- RISE Lab Deep Learning Overview:
 - https://ucbrise.github.io/cs294-rise-fa16/deep_learning.html
- [TensorFlow Python Tutorial](#)
- Stanford CS231 Labs
 - <http://cs231n.github.io/linear-classify/>
 - <http://cs231n.github.io/optimization-1/>
 - <http://cs231n.github.io/optimization-2/>

