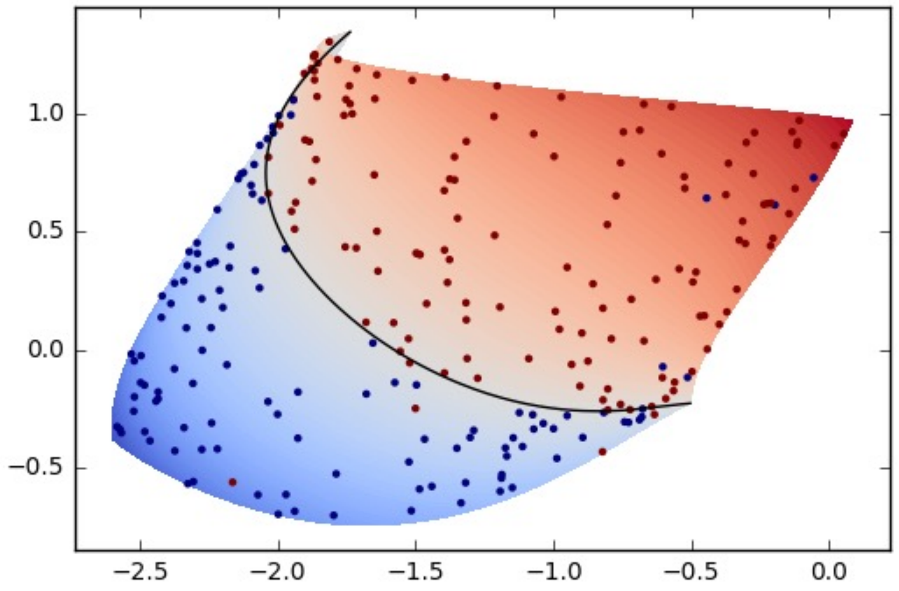
Lecture 4: Nonlinear Regression



Gavin Kerrigan

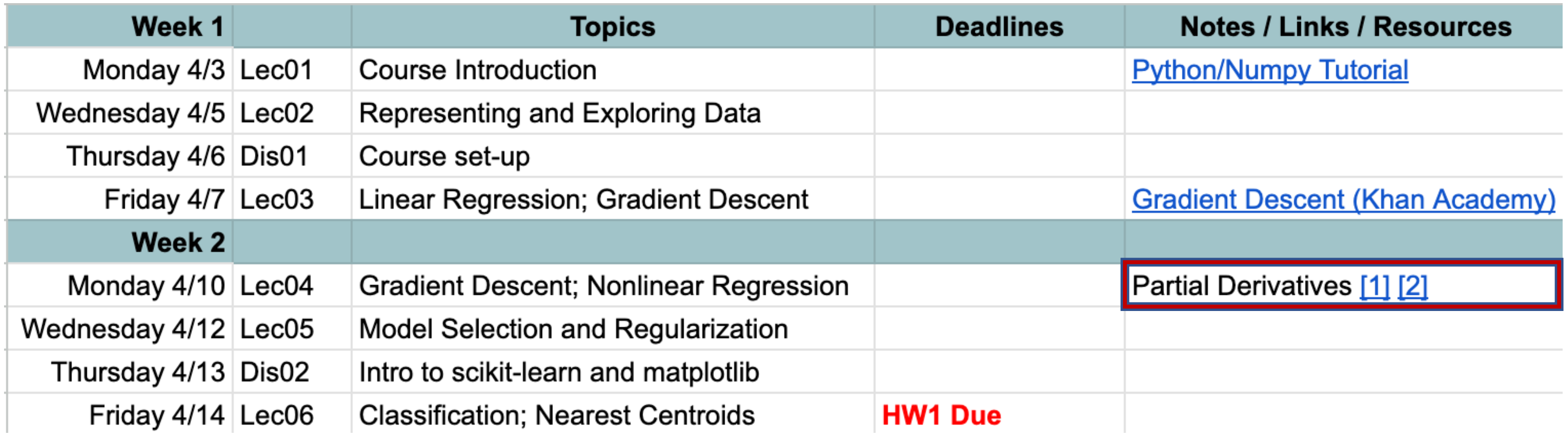
Spring 2023

Some materials courtesy Padhraic Smyth, Alex Ihler.

Announcements

- HW1 due Friday at Midnight
  - Post questions on Ed
  - Book TA office hours
- Problem 3: Nonlinear regression
  - Today's lecture
- Discussion section on Thursday
  - Matplotlib
  - Scikit-Learn
  - Bring questions about lecture/homework

Course Schedule



Week 1		Topics	Deadlines	Notes / Links / Resources
Monday 4/3	Lec01	Course Introduction		Python/Numpy Tutorial
Wednesday 4/5	Lec02	Representing and Exploring Data		
Thursday 4/6	Dis01	Course set-up		
Friday 4/7	Lec03	Linear Regression; Gradient Descent		Gradient Descent (Khan Academy)
Week 2				
Monday 4/10	Lec04	Gradient Descent; Nonlinear Regression		Partial Derivatives [1] [2]
Wednesday 4/12	Lec05	Model Selection and Regularization		
Thursday 4/13	Dis02	Intro to scikit-learn and matplotlib		
Friday 4/14	Lec06	Classification; Nearest Centroids	**HW1 Due**	

Questions?

Today’s Lecture

More on Gradient Descent

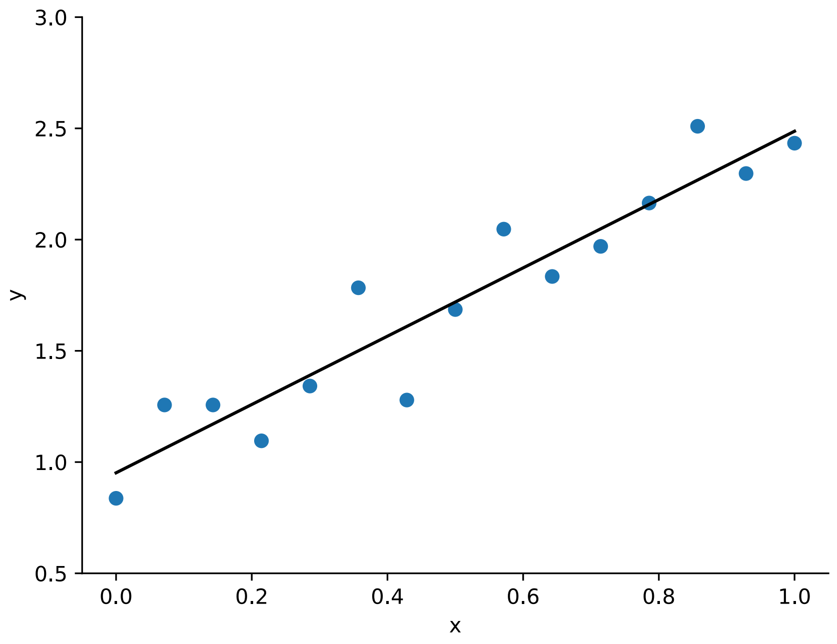
Nonlinear Regression

Model Selection

Reminders on Linear Regression

Linear regression model:

$$f(x \mid \theta) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = \theta^T \boldsymbol{x}$$



Reminders on Linear Regression

Linear regression model:

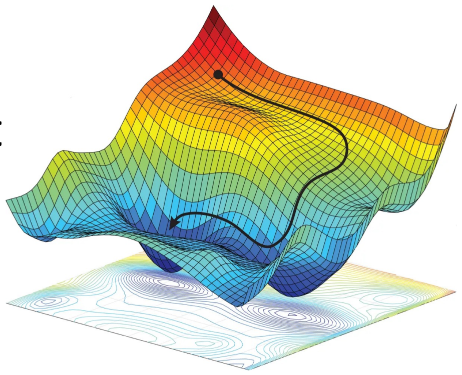
$$f(x \mid \theta) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = \theta^T \boldsymbol{x}$$

Model is learned by minimizing the MSE:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i|\theta))^2$$

MSE can be optimized via gradient descent

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta}^{old} - \lambda \cdot \nabla L(\boldsymbol{\theta}^{old})$$



Today's Lecture

More on Gradient Descent

Nonlinear Regression

Model Selection

Learning Rates

$$\theta^{new} = \theta^{old} - \lambda \cdot \nabla L(\theta^{old})$$

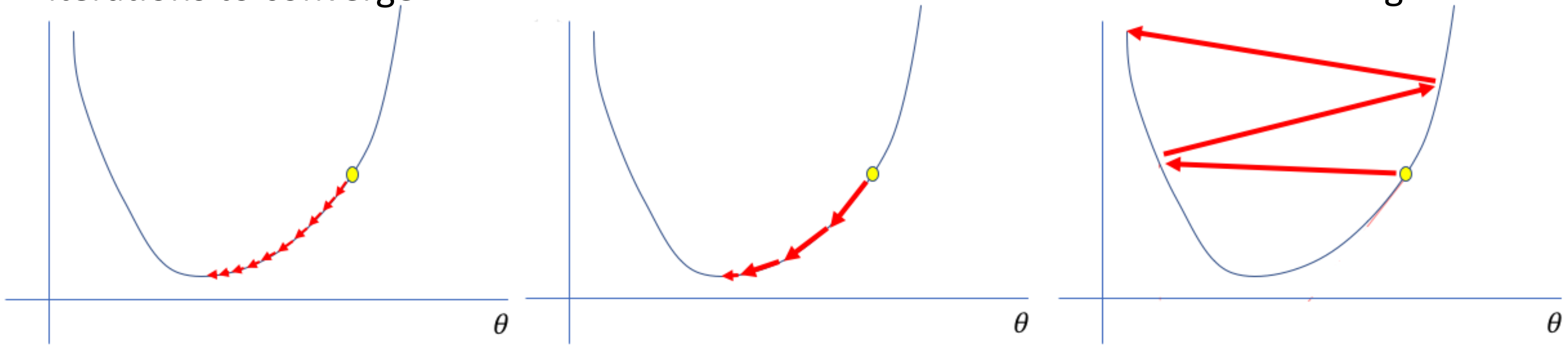
Learning rate (also known as stepsize)

How should we choose the learning rate?
λ is a "hyperparameter" – it is chosen by us, not learned
- Often one of the most important hyperparameters to tune

λ too small: requires many iterations to converge

λ just right

λ too large: may cause iterates to diverge



Jeremy Jordan

Learning Rates

$$\theta^{new} = \theta^{old} - \lambda \cdot \nabla L(\theta^{old})$$

Learning rate (also known as stepsize)

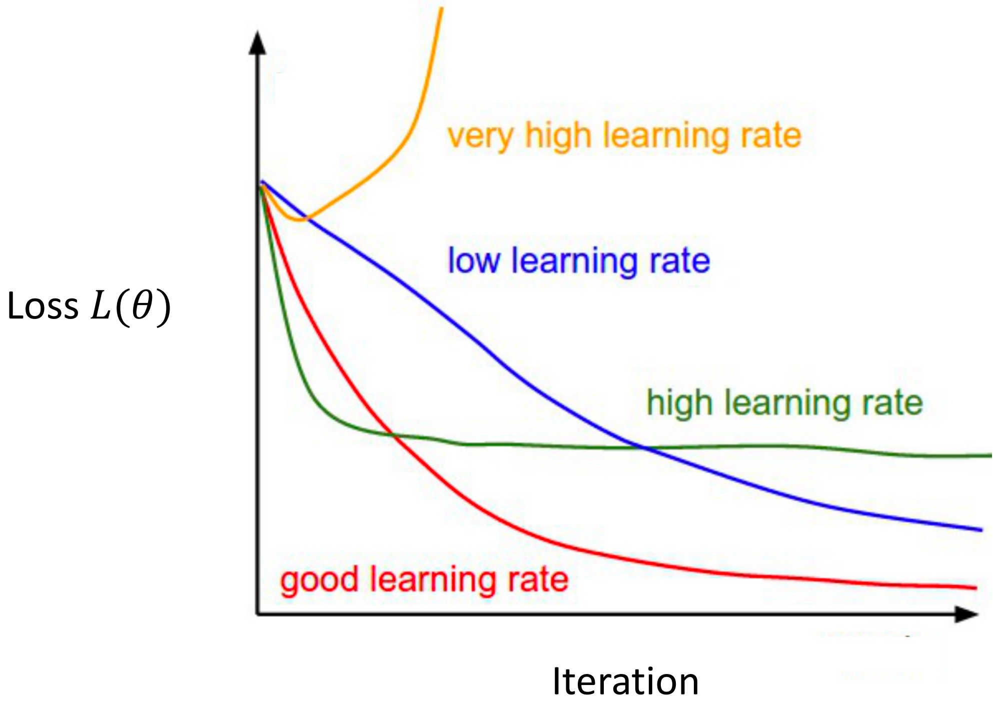
Some strategies for choosing λ:

- Train many models with different values of λ; choose the best model
    - Most straightforward option
    - Can be too costly if training models is expensive/slow

Learning Rates

$$\theta^{new} = \theta^{old} - \lambda \cdot \nabla L(\theta^{old})$$

Learning rate (also known as stepsize)



"Learning curves"

Stanford CS231n

Adaptive Learning Rates

$$\theta^{new} = \theta^{old} - \lambda \cdot \nabla L(\theta^{old})$$

Learning rate
(also known as
stepsize)

Some strategies for choosing λ:

- Use *adaptive* step sizes λ_t depending on the iteration count t
    - Idea: start out high to quickly get “close” to minimum; decrease over time to “fine tune”
    - e.g. *exponential decay:* $\lambda_t = e^{-kt}\,\lambda_0$

Decay rate $k > 0$

Initial learning rate λ_0

Adaptive Learning Rates

$$\theta^{new} = \theta^{old} - \lambda \cdot \nabla L(\theta^{old})$$

Learning rate (also known as stepsize)

Some strategies for choosing λ:

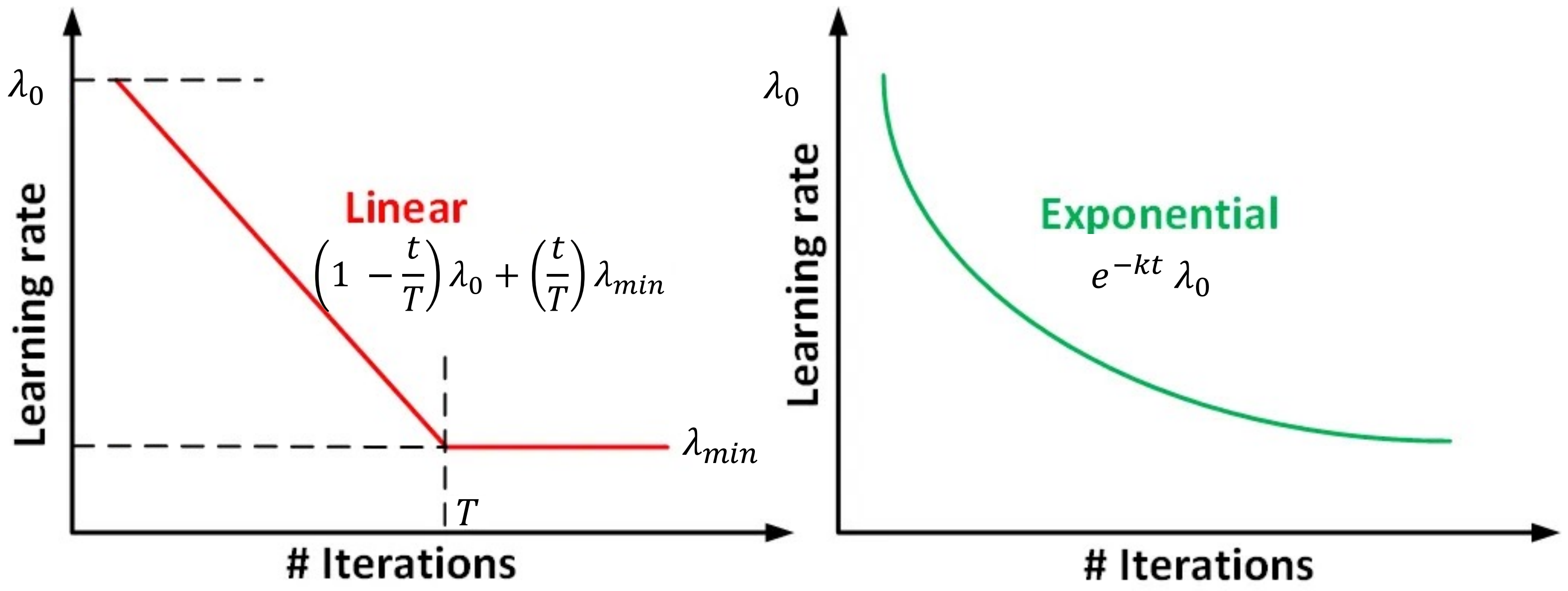
- Use *adaptive* step sizes λ_t depending on the iteration count t
  - Idea: start out high to quickly get "close" to minimum; decrease over time to "fine tune"
  - e.g. *linear decay:* $\lambda_t = \begin{cases} \left(1 - \frac{t}{T}\right)\lambda_0 + \left(\frac{t}{T}\right)\lambda_{min} & \text{if } t \leq T \\ \lambda_{min} & \text{if } t > T \end{cases}$

Number of decrease steps T

Initial learning rate λ_0

Minimal learning rate λ_{min}

Adaptive Learning Rates

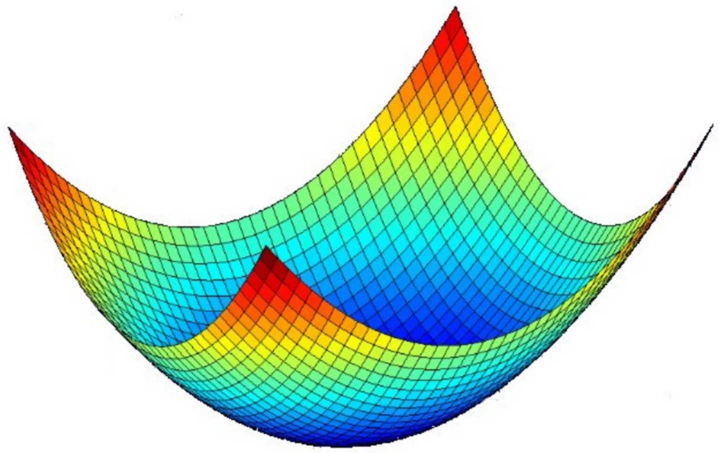


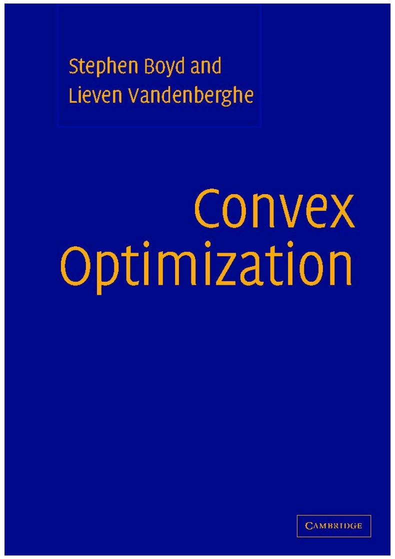
https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303

Initialization and Local Minima

For linear regression, the MSE loss function $L(\theta)$ is **<u>convex:</u>**

- Has a single, unique ("global") minimizer
- Gradient descent guaranteed to converge to the global minimizer (with appropriate choice of learning rate, plus other assumptions)
- Initialization of parameters affects how many iterations required


convex function


Stephen Boyd and
Lieven Vandenberghe

Convex
Optimization

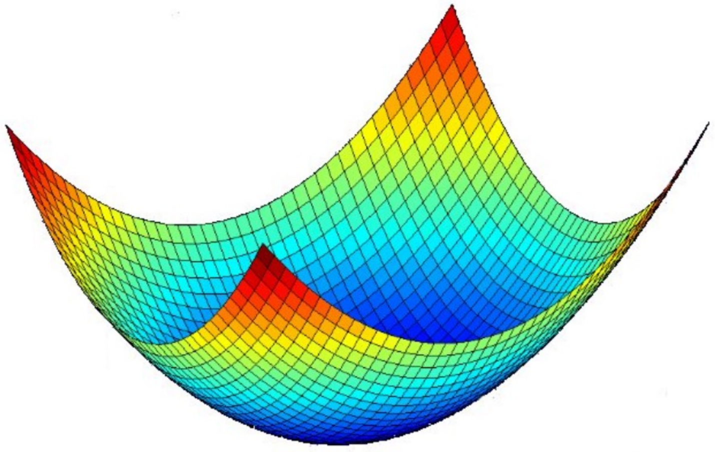
CAMBRIDGE

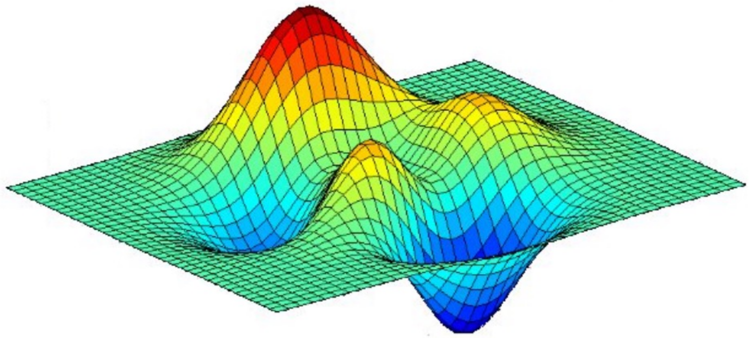
Kalin Kolev

Initialization and Local Minima

We'll see examples later of models with **non-convex** loss functions

- Gradient descent might only converge to a **local** minimum
- In general, no convergence guarantees for non-convex losses
- Sensitive to initialization of parameters


convex function

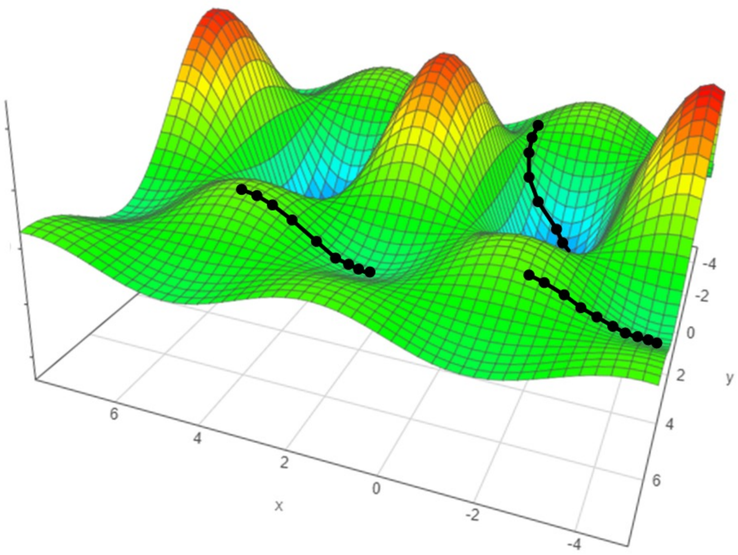

non-convex function

Kalin Kolev

Initialization and Local Minima

We'll see examples later of models with **non-convex** loss functions

- Gradient descent might only converge to a **local** minimum

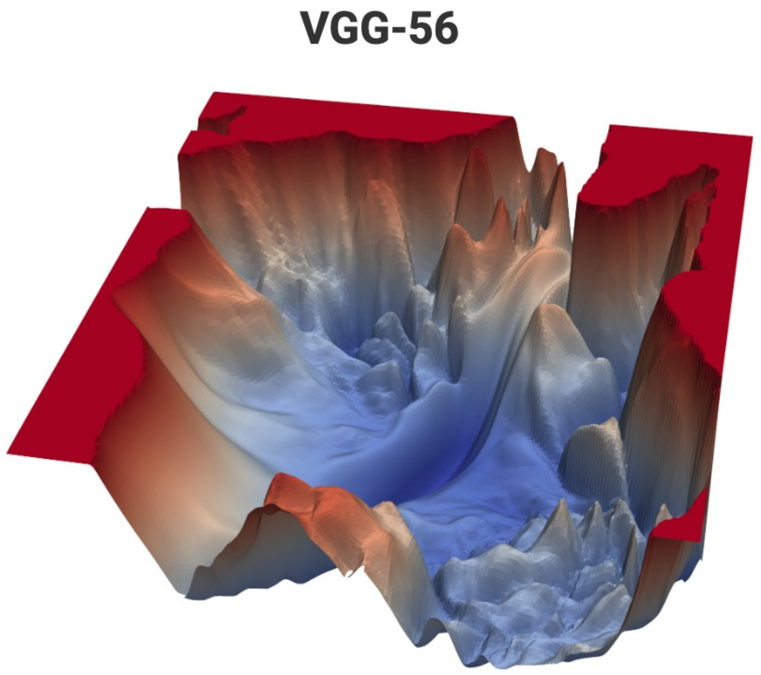


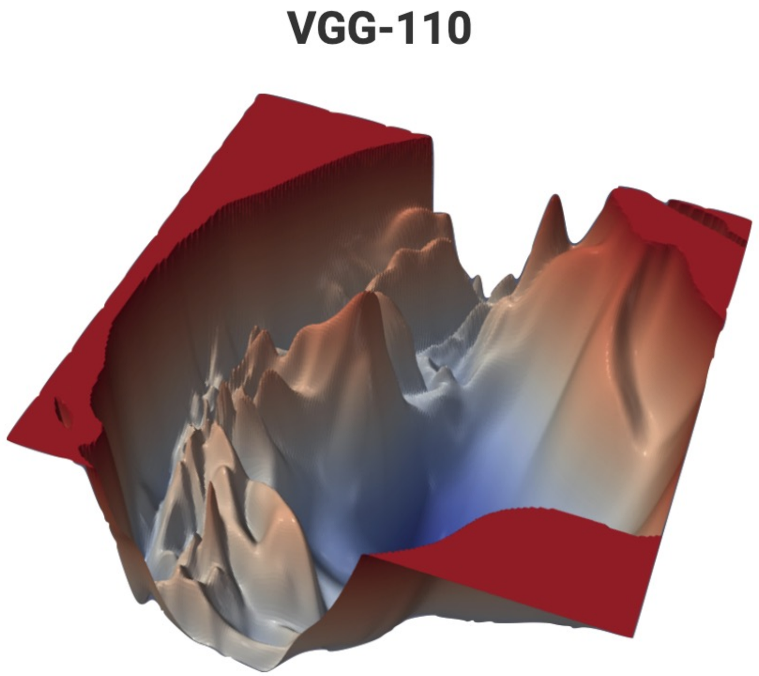
http://www.offconvex.org/2018/11/07/optimization-beyond-landscape/

Initialization and Local Minima

We'll see examples later of models with **non-convex** loss functions

- Gradient descent might only converge to a **local** minimum
- E.g. neural networks can have very complicated loss surfaces





Hao Li et al. 2017, Visualizing the Loss Landscape of Neural Nets, losslandscape.com

Feature Scaling

For any gradient-based learning algorithm its useful for features to be on the same scale

Standard Scaling Method:

For each feature

- subtract the mean (results in a feature with 0 mean)
- divide by the standard deviation

$$x_{ij} \leftarrow \frac{x_{ij} - \overline{x_j}}{\sigma_j}$$

$\overline{x_j}, \sigma_j$ are the mean/standard deviation of feature j

Results in each feature having mean 0 and standard deviation 1 (i.e., all features are on the same scale)

Feature Scaling

For any gradient-based learning algorithm its useful for features to be on the same scale

Standard Scaling Method:

For each feature

- subtract the mean (results in a feature with 0 mean)
- divide by the standard deviation

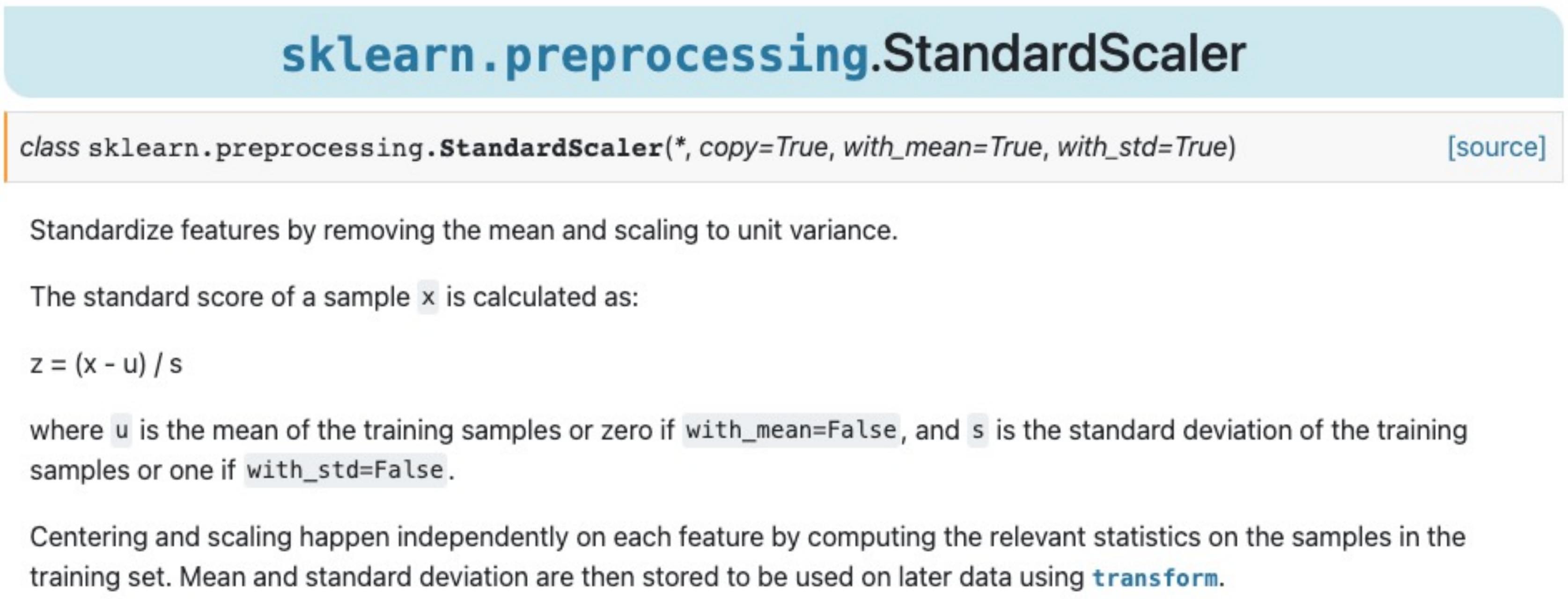
$$x_{ij} \leftarrow \frac{x_{ij} - \overline{x_j}}{\sigma_j}$$

$\overline{x_j}, \sigma_j$ are the mean/standard deviation of feature j

Why useful for gradient-based methods?

If one feature is much larger than others,

it will have an oversize impact on the gradient



sklearn.preprocessing.StandardScaler

class `sklearn.preprocessing.StandardScaler`(**, copy=True, with_mean=True, with_std=True*) [source]

Standardize features by removing the mean and scaling to unit variance.

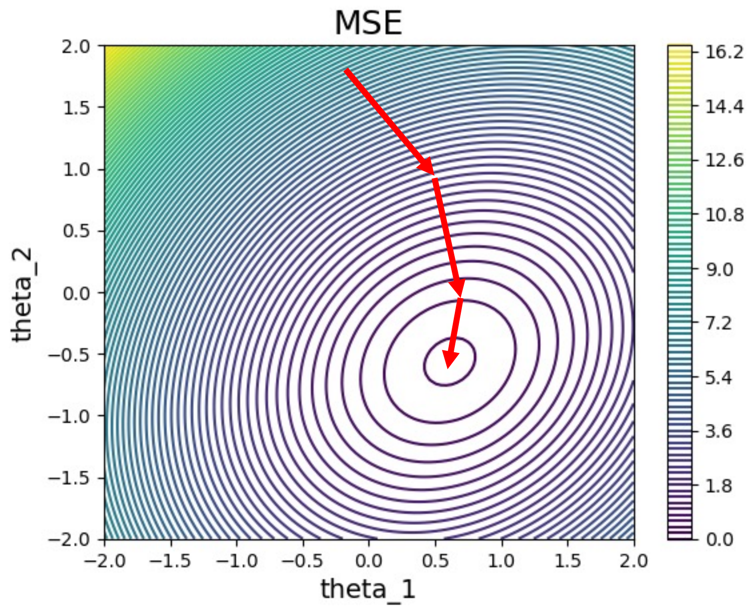
The standard score of a sample `x` is calculated as:

z = (x - u) / s

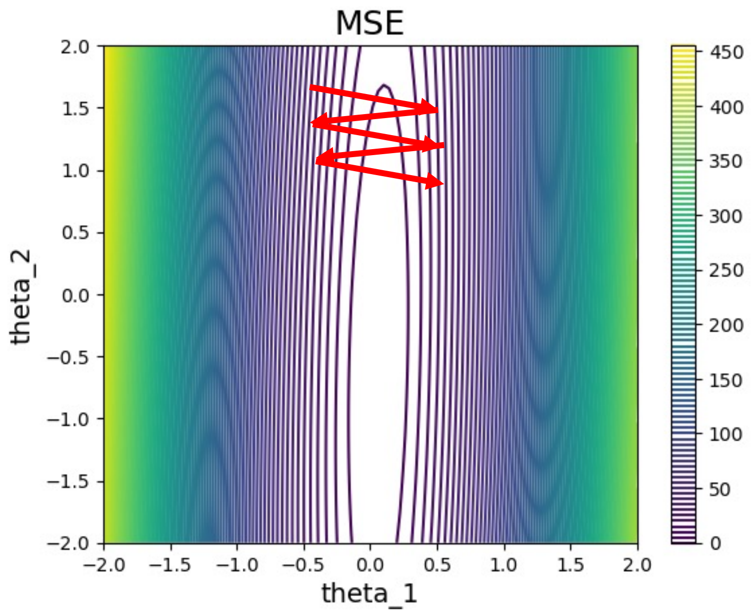
where `u` is the mean of the training samples or zero if `with_mean=False`, and `s` is the standard deviation of the training samples or one if `with_std=False`.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using `transform`.

Effect of Scaling on Gradients



Loss surface (for MSE, linear model) where features have been scaled



Loss surface (for MSE, linear model) where feature 1 has standard deviation = 5, and other variables are standardized (std = 1)

Methods for Detection of Convergence

Method 1: Magnitude of gradient goes to 0

e.g., sum of absolute values of gradient vector are < small threshold

Method 2: Loss stops changing

e.g., change in loss over last 2 iterations < small threshold

Method 3: Parameters stop changing

e.g., change in parameters over last 2 iterations < small threshold

Methods 1 or 2 used in practice

Alternative (later): monitor loss on a validation dataset

Variations on Gradient Descent

- Many variations on the basic gradient descent idea
  - Optimization is very widely used in training ML models, so there is extensive research (ongoing) on improving basic methods
- Momentum methods:
  - New direction = a weighted average of the current gradient with the previous gradient
  - Can smooth out gradient steps, can be faster (mileage will vary)
- 2nd-order (Newton) methods:
  - Use 2nd-order derivatives (as well as 1st order)
  - Scales as $O(d^3)$, not widely used in machine learning

Questions?

Today’s Lecture

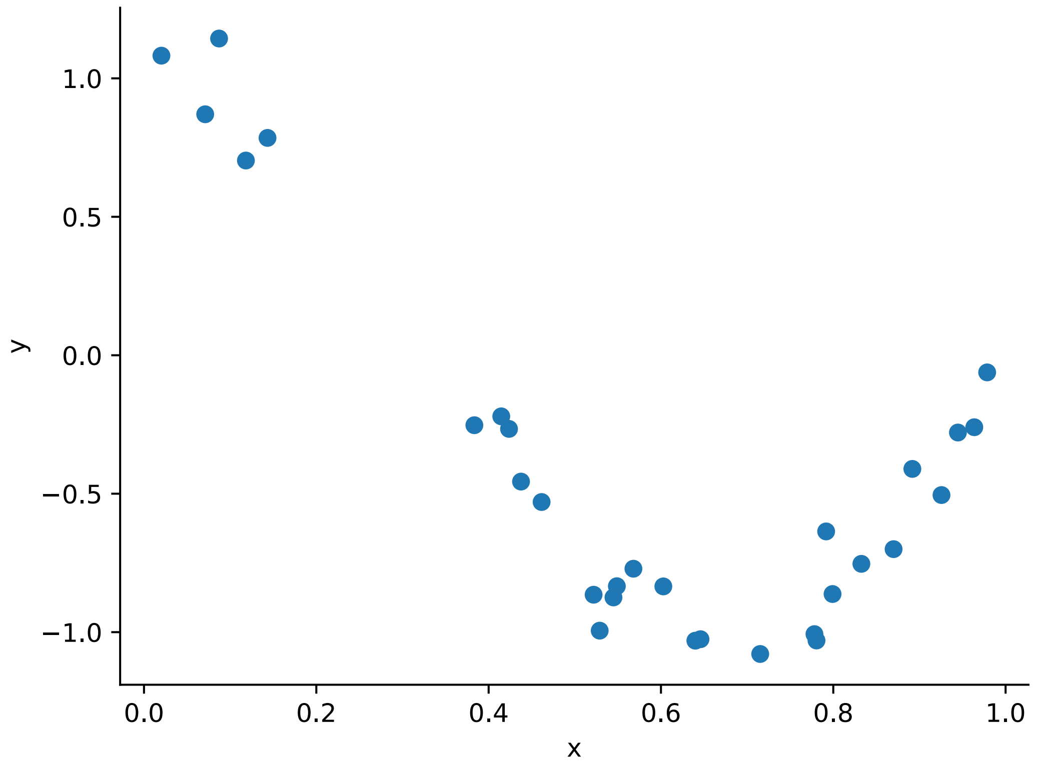
More on Gradient Descent

Nonlinear Regression

Model Selection

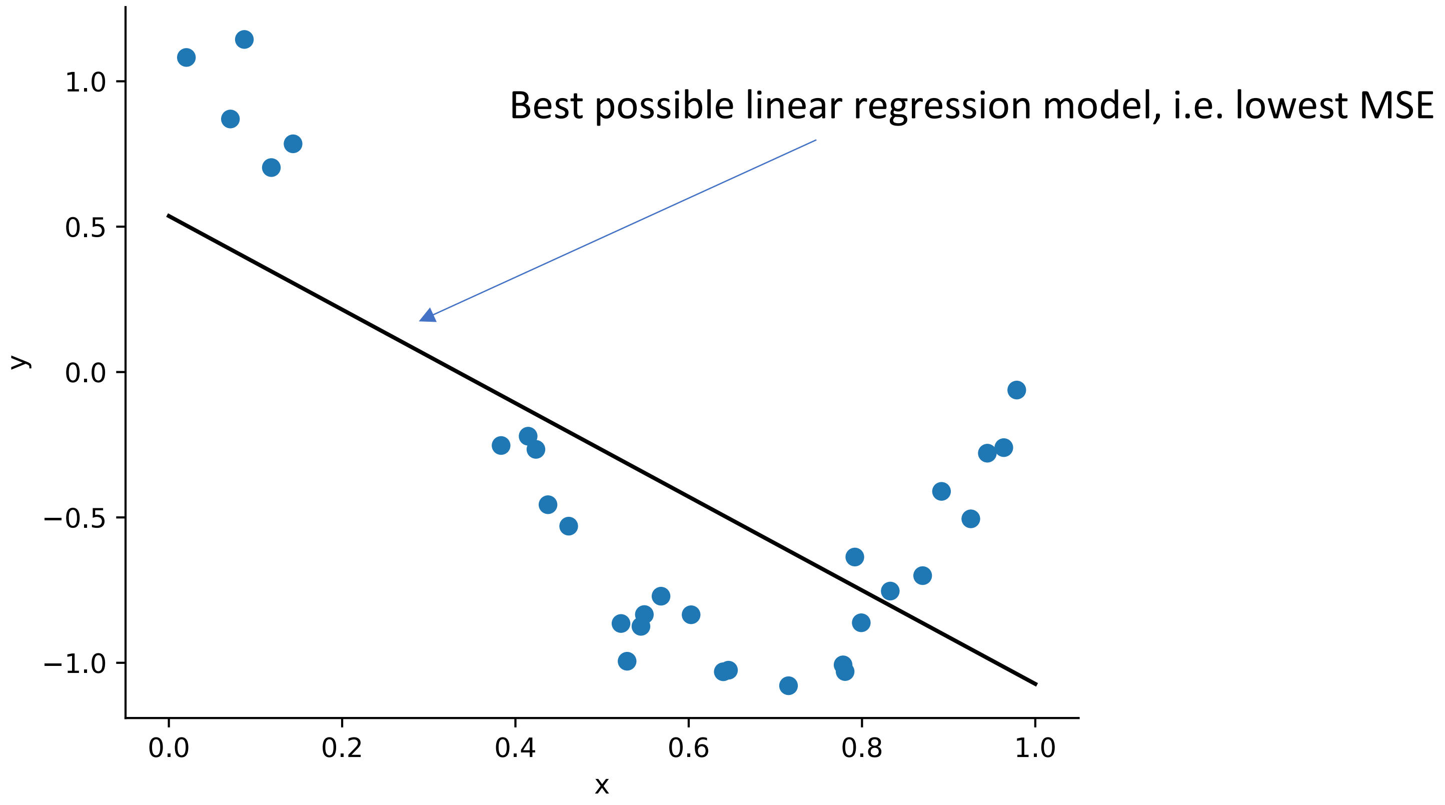
Nonlinear Regression

How can we do regression if our data isn't (approximately) linear?



Nonlinear Regression

How can we do regression if our data isn't (approximately) linear?



Polynomial Feature Expansions

Suppose data only has one feature x_1

Linear regression model can (unsurprisingly) only model linear relationships

$$f(x \mid \theta) = \theta_0 + \theta_1 x_1$$

<u>Quadratic</u> regression can model more complex relationships

- No longer line of best fit but "parabola of best fit"

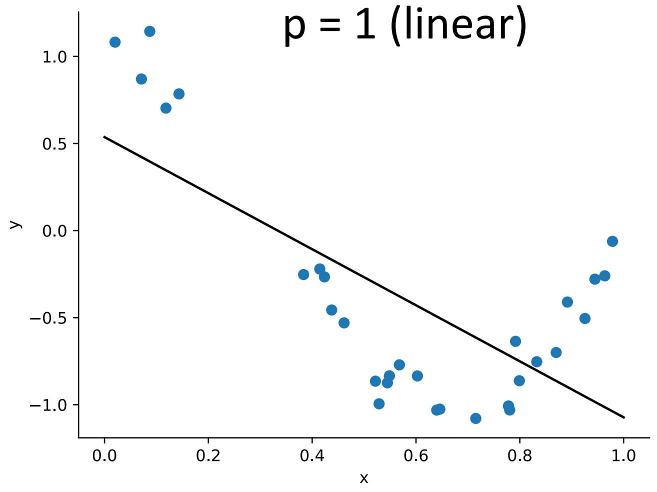
$$f(x \mid \theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

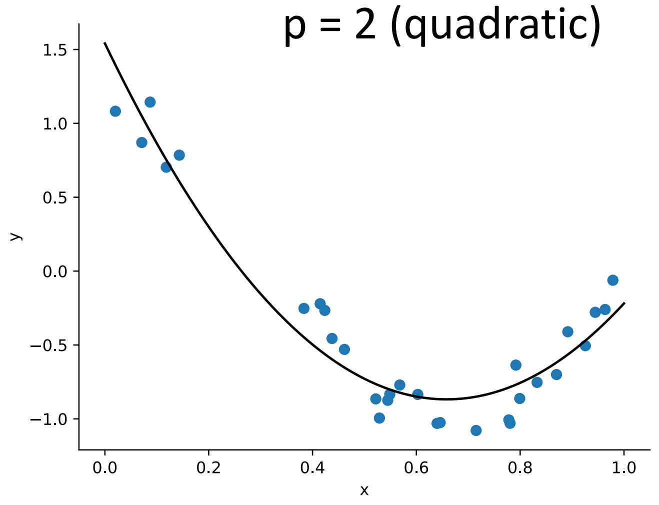
<u>Polynomial</u> regression models are even more flexible:

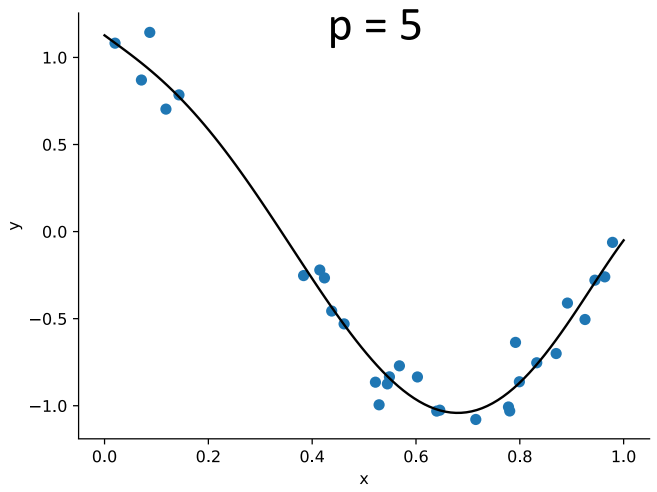
p is the **<u>degree</u>** of the polynomial expansion

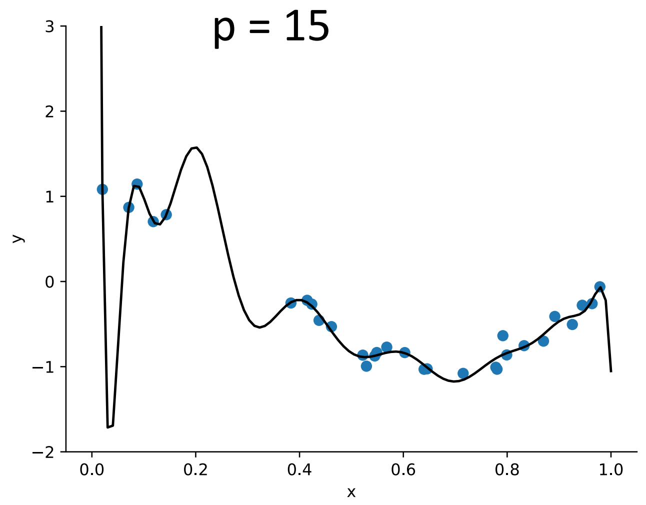
$$f(x \mid \theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \cdots + \theta_p x_1^p$$

Polynomial Feature Expansions









Polynomial Feature Expansions

How can we find the polynomial of best fit?

- Fix a degree p – a *hyperparameter* of our model

Consider now a feature vector with d features

$$\mathbf{x} = (x_1, x_2, \ldots, x_d)$$

The **polynomial feature expansion** of $\boldsymbol{x}$ in degree 2 is

$$\boldsymbol{z} = \underbrace{\phi_2(\boldsymbol{x})}_{\text{“feature map”}} = (1, x_1, x_2, \ldots, x_d, x_1^2, x_2^2, \ldots, x_d^2, x_1x_2, x_1x_3, \ldots, x_2x_3, \ldots)$$

Key idea: we are creating a *new* feature vector **z** consisting of all first and second-degree feature interactions

Polynomial Feature Expansions

How can we find the polynomial of best fit?

- Fix a degree p – a *hyperparameter* of our model

Consider now a feature vector with d features

$$\mathbf{x} = (x_1, x_2, \ldots, x_d)$$

The **polynomial feature expansion** of $\boldsymbol{x}$ in degree p is

$$\boldsymbol{z} = \phi_p(\boldsymbol{x}) = (1, x_1, x_2, \ldots, x_d, x_1^2, x_2^2, \ldots, x_d^2, \ldots, x_1^p, \ldots, x_d^p, \ldots, x_2 x_3^{p-1}, \ldots)$$

New feature vector consisting of all possible feature combinations of degree $\leq p$

Polynomial Feature Expansions

Example of polynomial feature expansions:

- Feature vector with $d = 3$ features

$$\boldsymbol{x} = (1, x_1, x_2, x_3)$$

Degree $p = 2$ (quadratic) expansion:

$$\boldsymbol{z} = \phi_2(\boldsymbol{x}) = (1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3)$$

- All possible feature combinations of degree at most 2

Check your understanding: compute $\boldsymbol{z} = \phi_3(\boldsymbol{x})$

Polynomial Regression

How do we actually fit a polynomial to data?

- Suppose our data only has a single feature x_1

Degree-p polynomial regression model:

$$f(\, x \mid \theta \,) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \cdots + \theta_p x_1^p$$

Equivalent to doing a degree-p feature expansion followed by *linear regression*:

$$\boldsymbol{z} = \phi_p(x) = \left(1, x_1, x_1^2, \dots, x_1^p\right) = \left(z_0, z_1, z_2, \dots, z_p\right)$$

$$f(\, \boldsymbol{z} \mid \theta \,) = \theta_0 + \theta_1 z_1 + \theta_2 z_2 + \cdots + \theta_p z_p = \sum_{k=0}^{p} \theta_k z_k$$

Polynomial Regression

How do we actually fit a polynomial to data?

- Suppose our data only has a single feature x_1

Equivalent to doing a degree-p feature expansion followed by *linear regression*:

$$\boldsymbol{z} = \phi_p(x) = \left(1, x_1, x_1^2, \dots, x_1^p\right) = \left(z_0, z_1, z_2, \dots, z_p\right)$$

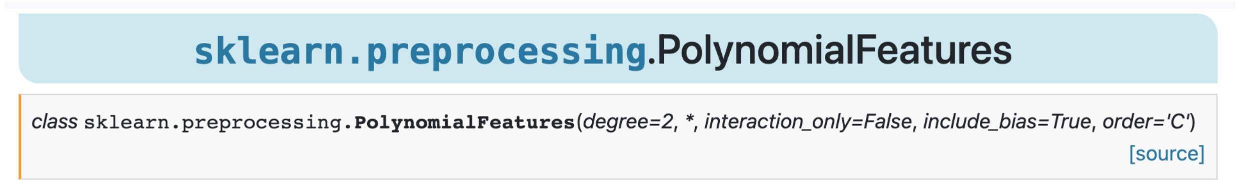
$$f(\,\boldsymbol{z} \mid \theta\,) = \theta_0 + \theta_1 z_1 + \theta_2 z_2 + \cdots + \theta_p z_p = \sum_{k=0}^{p} \theta_k z_k$$

- Same idea works for data with more than one feature
- **Key idea**: polynomial regression is just linear regression with new features

Polynomial Regression

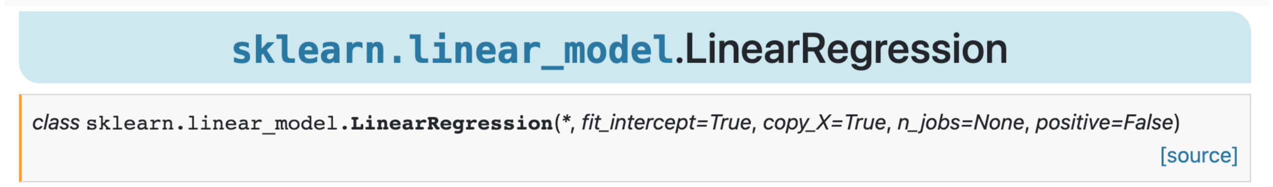
Fitting a polynomial to data:

1. Fix a polynomial degree p
2. Compute the feature expansions $\boldsymbol{z}_i = \phi_p(\boldsymbol{x}_i)$ for every datapoint $i = 0, 1, 2, \dots, n$


sklearn.preprocessing.PolynomialFeatures

class `sklearn.preprocessing.PolynomialFeatures`(*degree=2, *, interaction_only=False, include_bias=True, order='C'*)
[source]

3. Fit a linear regression model with the *new features* $\boldsymbol{z}_i$

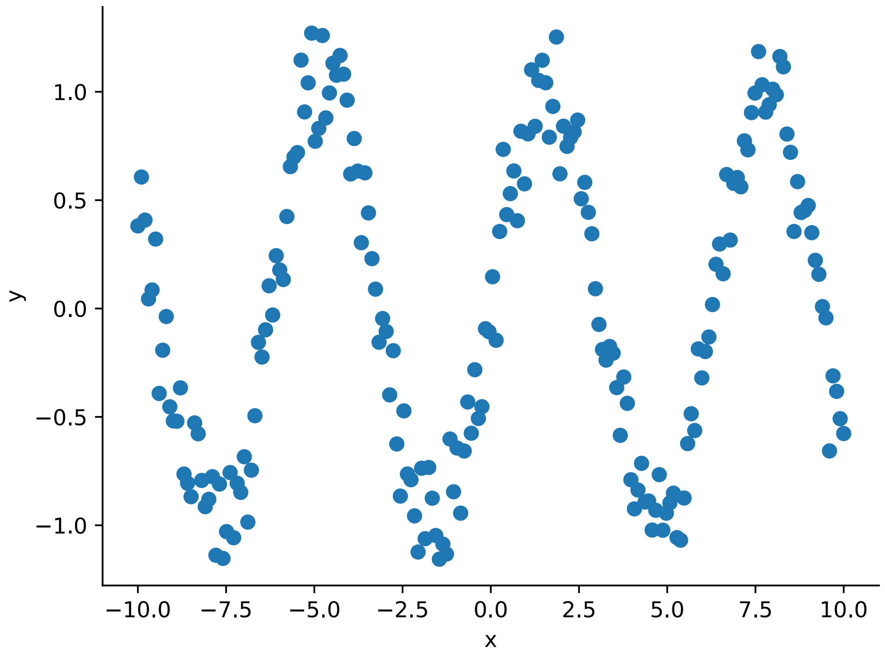

sklearn.linear_model.LinearRegression

class `sklearn.linear_model.LinearRegression`(**, fit_intercept=True, copy_X=True, n_jobs=None, positive=False*)
[source]

Questions?

Nonlinear Regression

What would you do if your data looked like this?

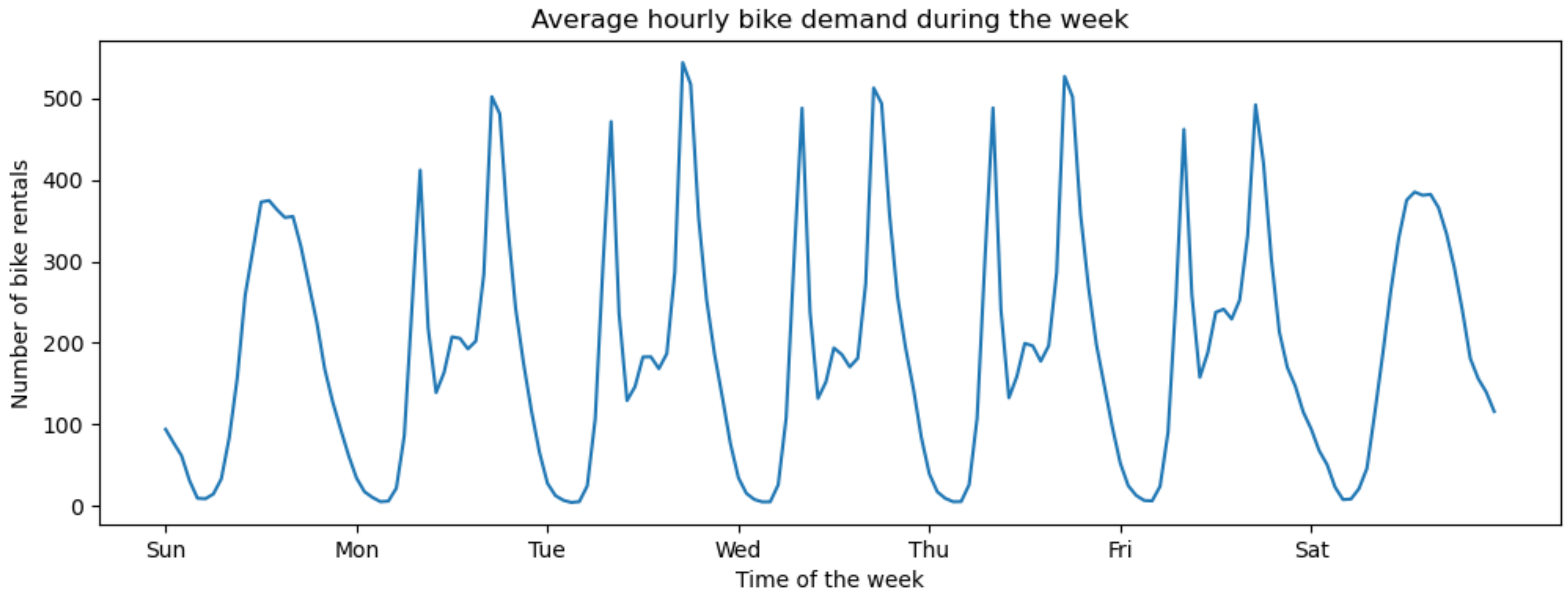


Feature expansions aren't limited to polynomials! E.g.:

$$\boldsymbol{z} = \phi_{trig}(\boldsymbol{x}) = (1, x_1, \sin(x_1), \cos(x_1), \sin(2x_1), \cos(2x_1), \dots)$$

Nonlinear Regression

What would you do if your data looked like this?



https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html

Designing feature maps is often called “feature engineering”

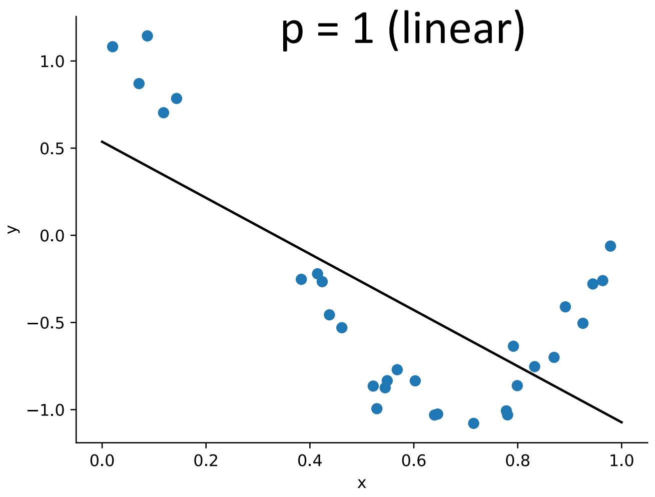
Today’s Lecture

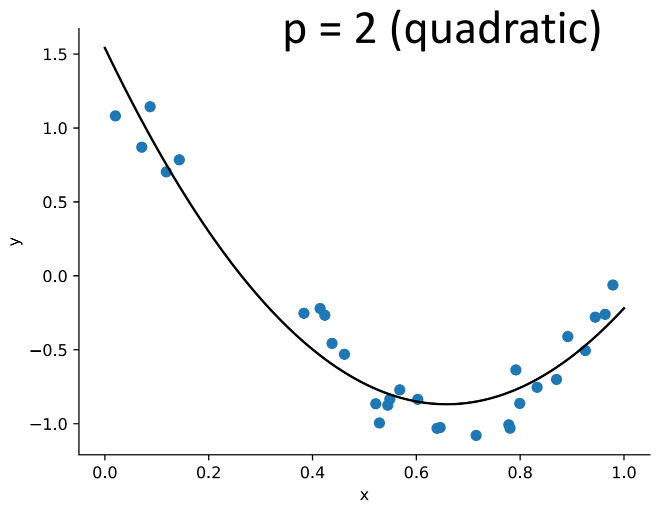
More on Gradient Descent

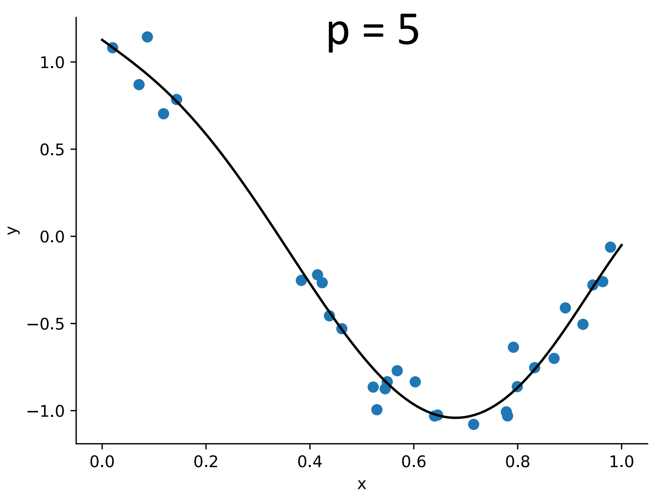
Nonlinear Regression

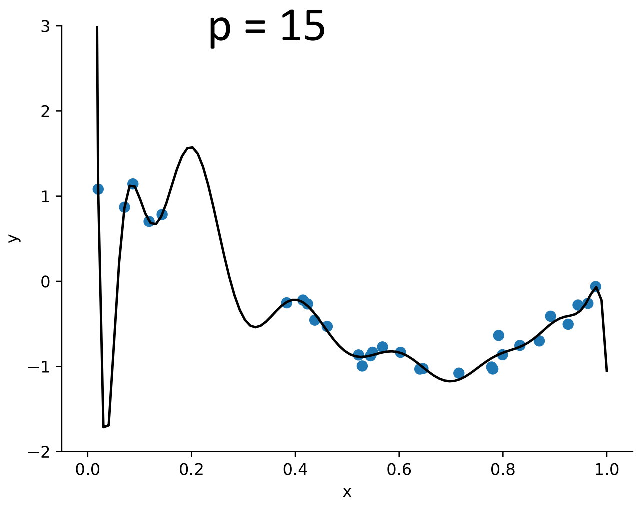
Model Selection

What polynomial degree should we use?







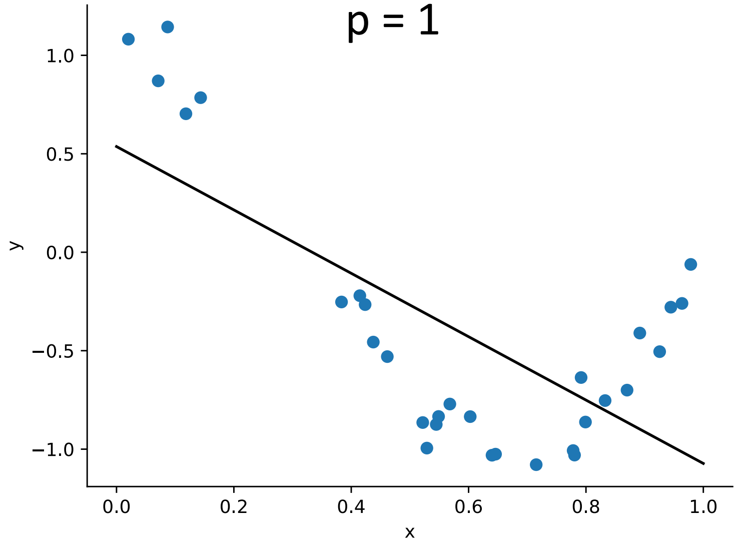


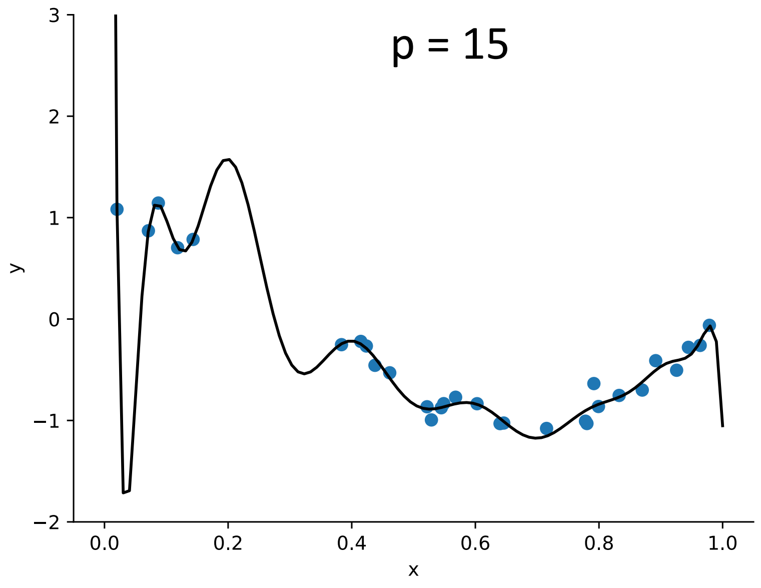
Overfitting and Underfitting

The complexity of our model should “match” the complexity of our data

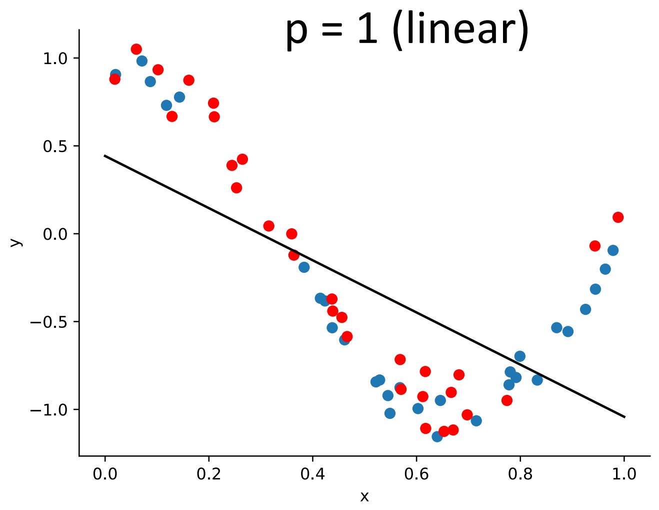
Underfitting: model is too simple to appropriately capture the data

Overfitting: model is too complex; fits data too well and cannot generalize to data not seen during training





What polynomial degree should we use?

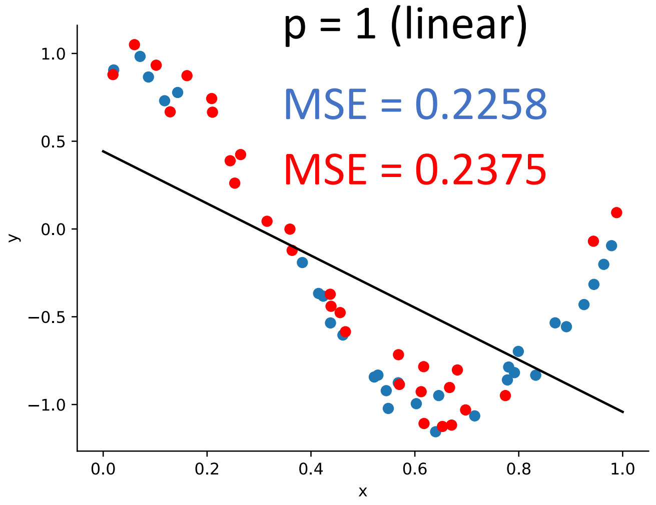


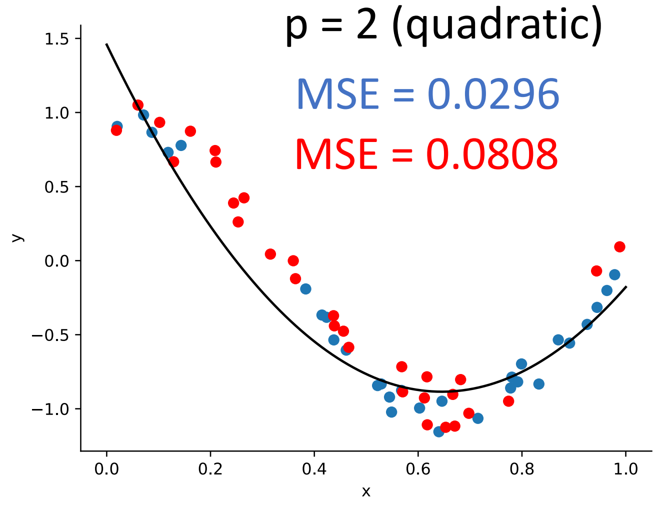
Blue dots: training data, i.e. model is fit using this data

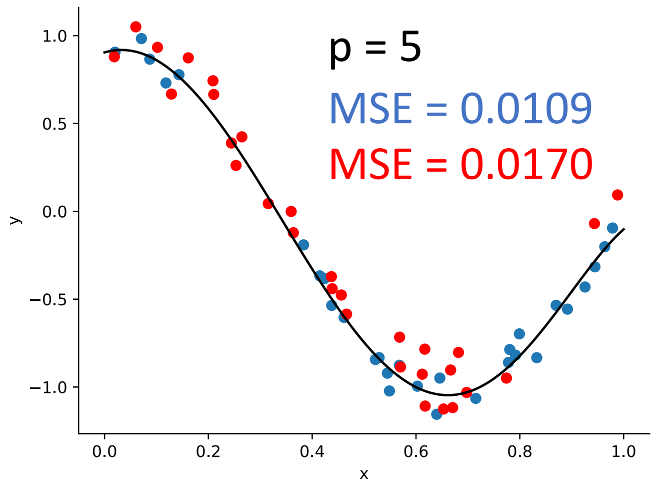
Red dots: validation data, only seen *after* we have trained our model

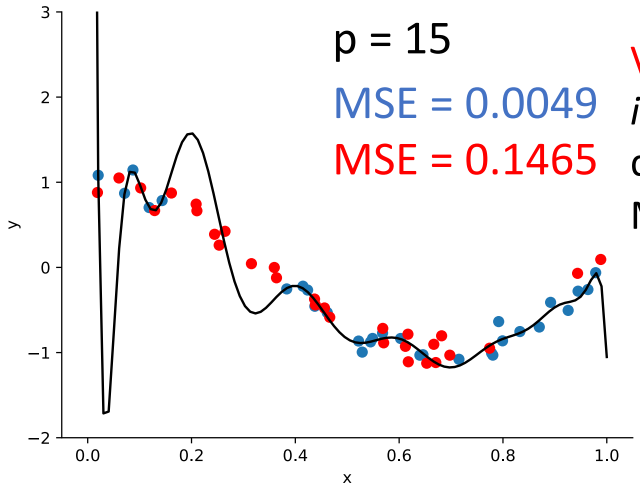
- e.g. we deployed our model out in the world and are using it to make predictions for the red points

What polynomial degree should we use?









Validation MSE *increases* despite training MSE decreasing

Overfitting and Underfitting

The complexity of our model should “match” the complexity of our data

Underfitting: model is too simple to appropriately capture the data

Overfitting: model is too complex; fits data too well and cannot generalize to data not seen during training

One method for detecting over/under fitting is to *partition* your data:

1. Training subset: model is fit on this data
2. Validation subset: *held-out* data not seen by the model during training
   - Lets you evaluate the model on unseen data
   - Used for model selection / tuning hyperparameters
3. Testing subset:
   - Used to assess model generalization *after model selection*
   - (more on this next lecture)

Overfitting and Underfitting

How can we determine if a model is overfitting or underfitting?

- A heuristic (but not set-in-stone) rule:

	High Training Error	Low Training Error
High Val. Error	**Underfitting**	**Overfitting**
Low Val. Error	Bug in your code ☺ (or train/val mismatch)	Appropriate Fit

Summary and Wrapup

- More details on gradient descent
  - Learning rates
  - Convexity
  - Feature scaling

- Nonlinear regression
  - Feature expansions: polynomial, trigonometric, …
  - Can be seen as linear regression with *new* features

- Model selection basics:
  - Overfitting and Underfitting

Next Lecture

- More on model selection
  - Train/Val/Test splits
  - Cross-Validation
  - Bias-Variance Tradeoff

- Methods for preventing overfitting
  - “Regularization”

Questions?
Outside after lecture