CS 4342: Class 6

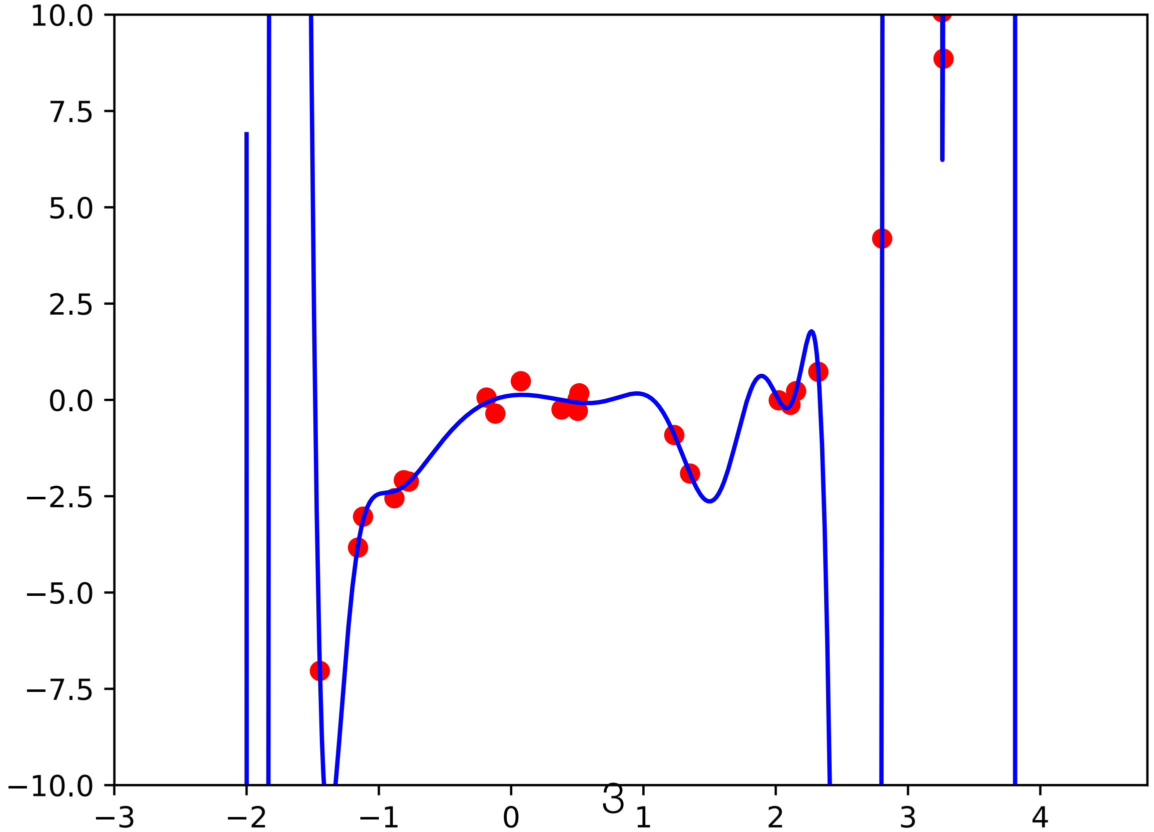
Jacob Whitehill

Overfitting and regularization

Overfitting

- If polynomial regression with degree 3 worked well, why not increase the degree even higher?

- Let's try with degree 25…



We nailed almost every point exactly!… but maybe this is overkill?

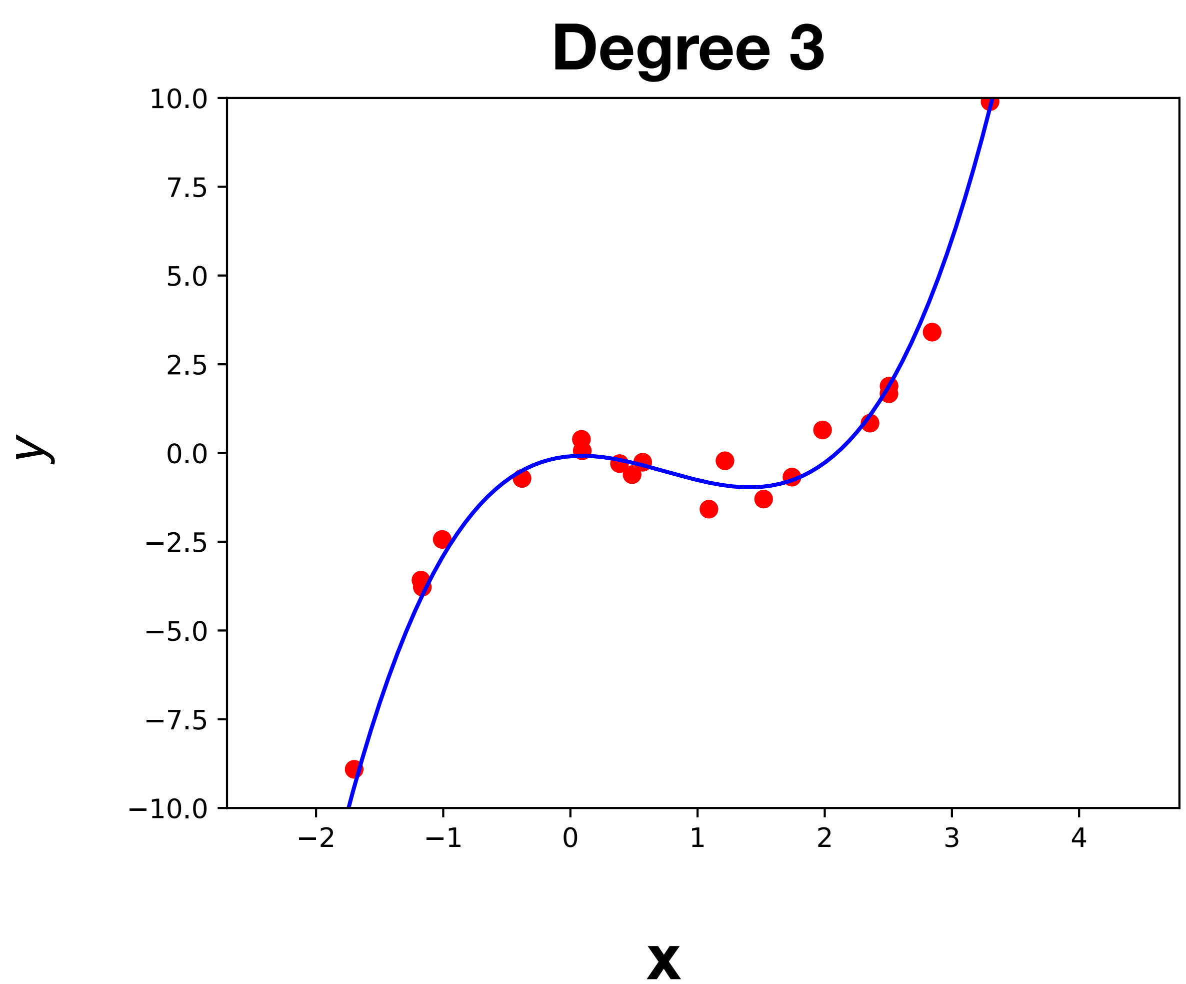
Overfitting

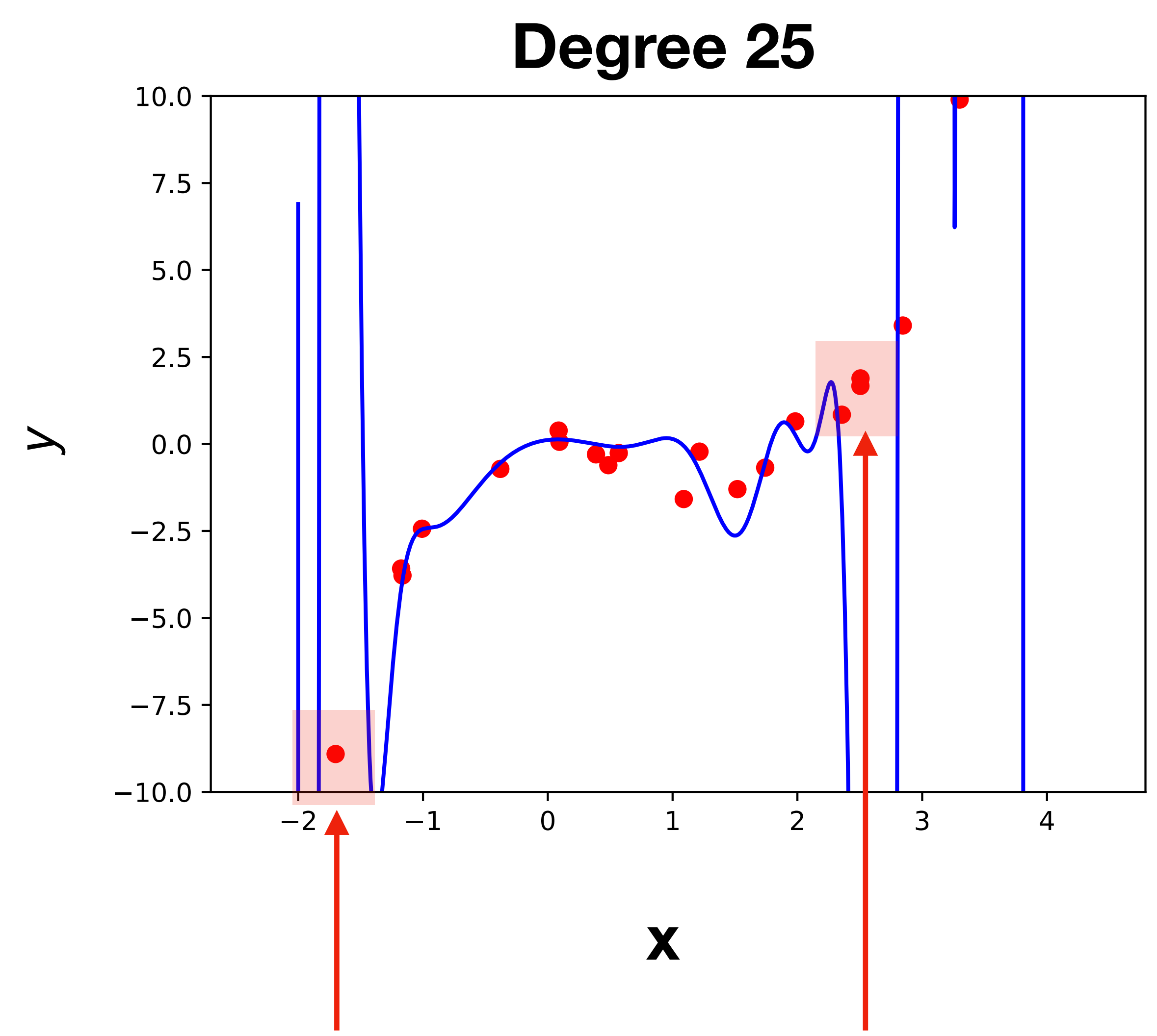
- Why is this bad? Recall that **overfitting** means that training error is low, but testing error is high.

- **Testing error** represents how well we expect our machine to perform on data we have **not seen before**.

Overfitting

- Here are the machine's predictions using polynomial regression, with either degree 3 or degree 25:





For these data points, the predictions are very inaccurate, which makes f_{MSE} large.

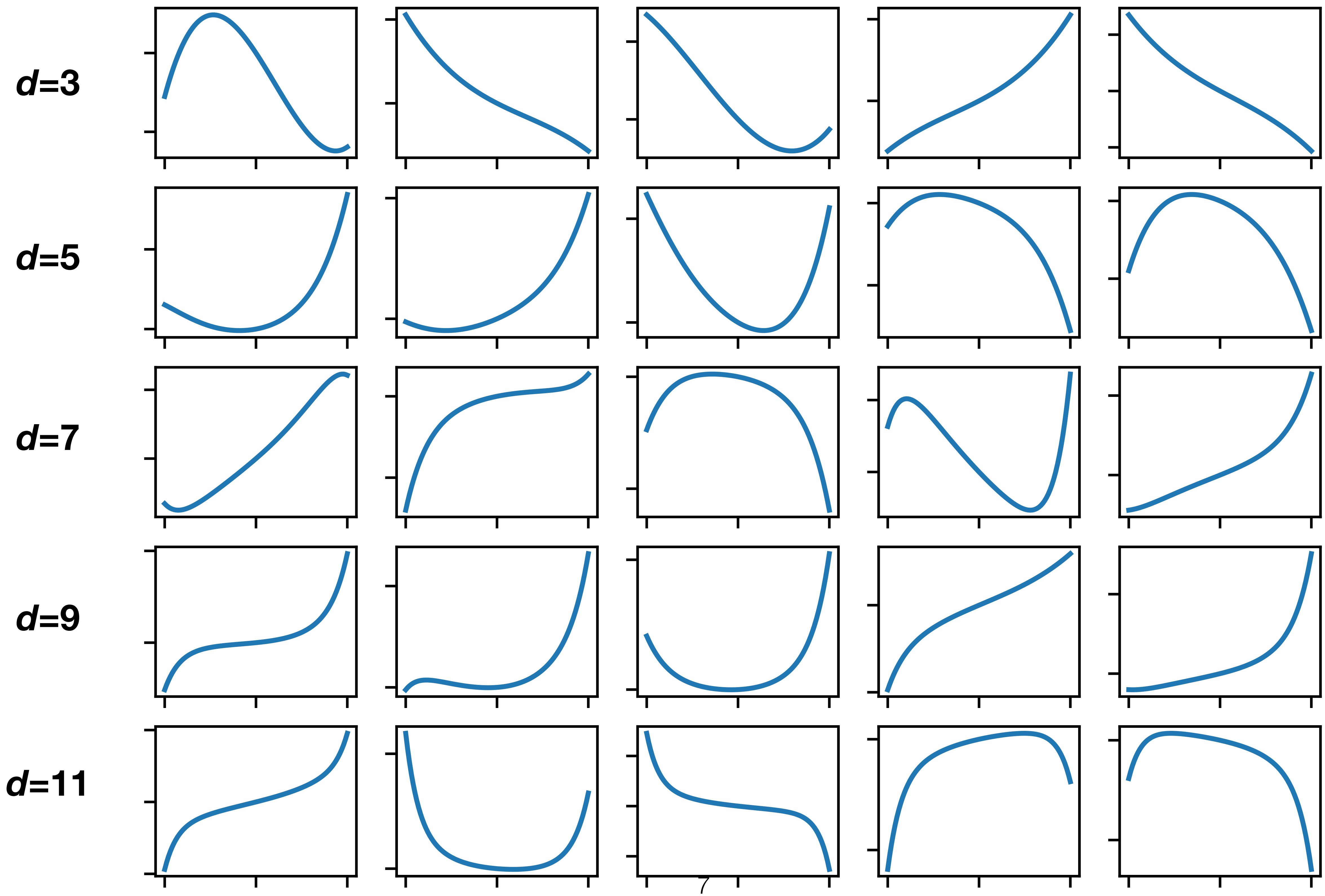
Preventing overfitting

- How to prevent this? Two strategies:
  - Keep the *degree d* of the polynomial modest.
  - Keep the *weight* associated with each term modest.

$$\hat{y} = w_0 x^0 + w_1 x^1 + w_2 x^2 + \ldots + w_d x^d$$

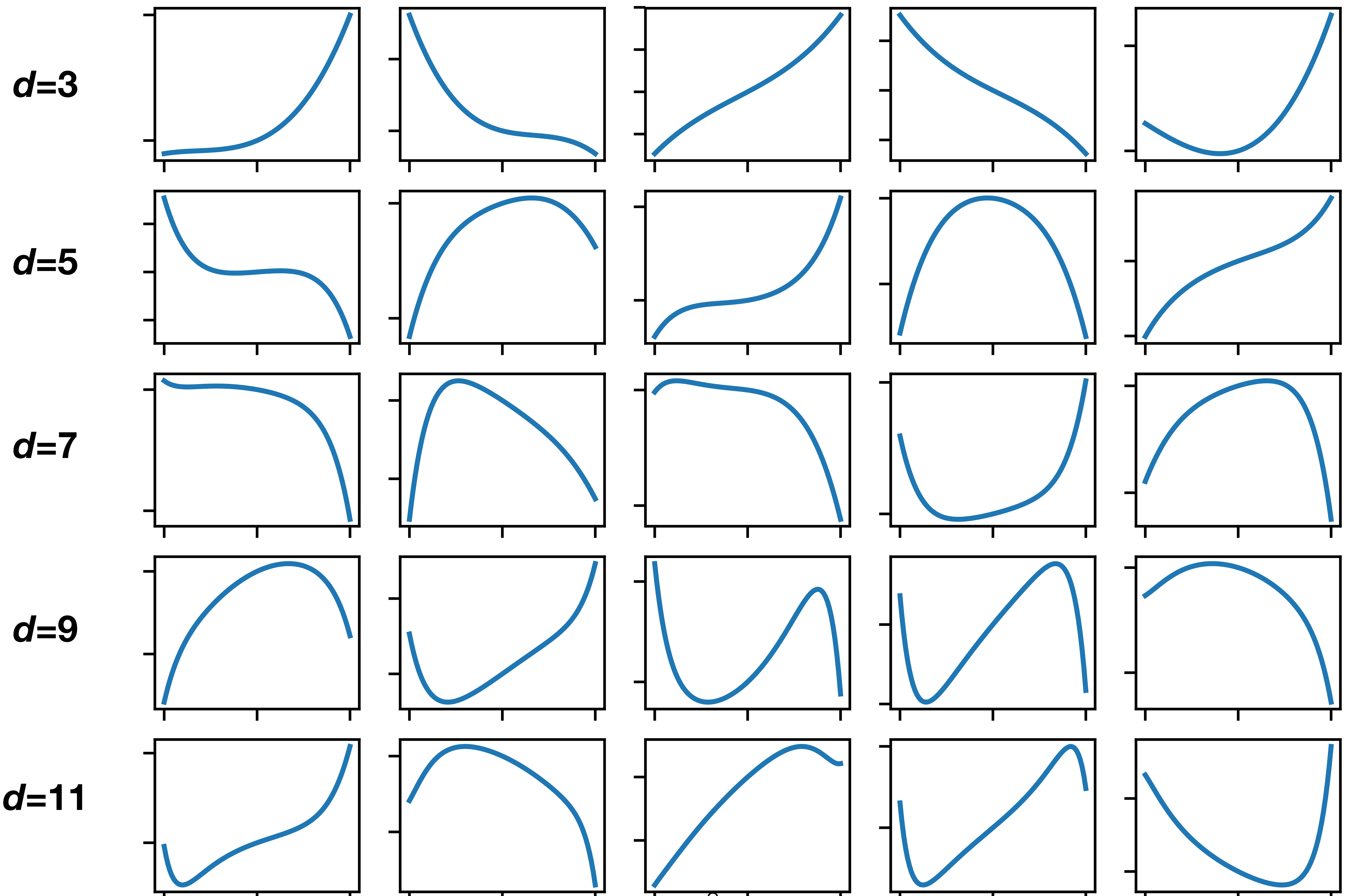
Random polynomials

μ=7.25e-07



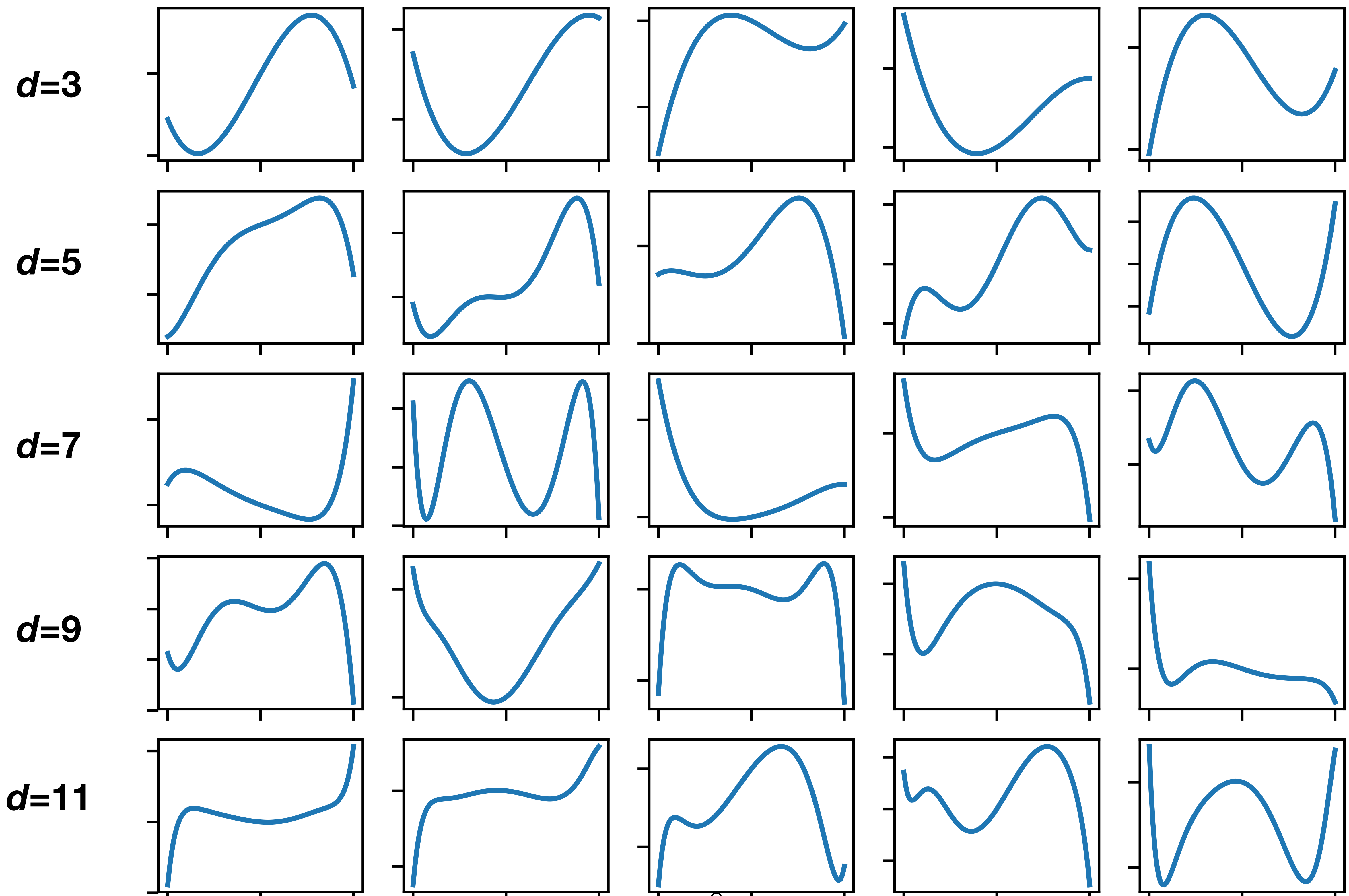
Random polynomials

μ=0.00063

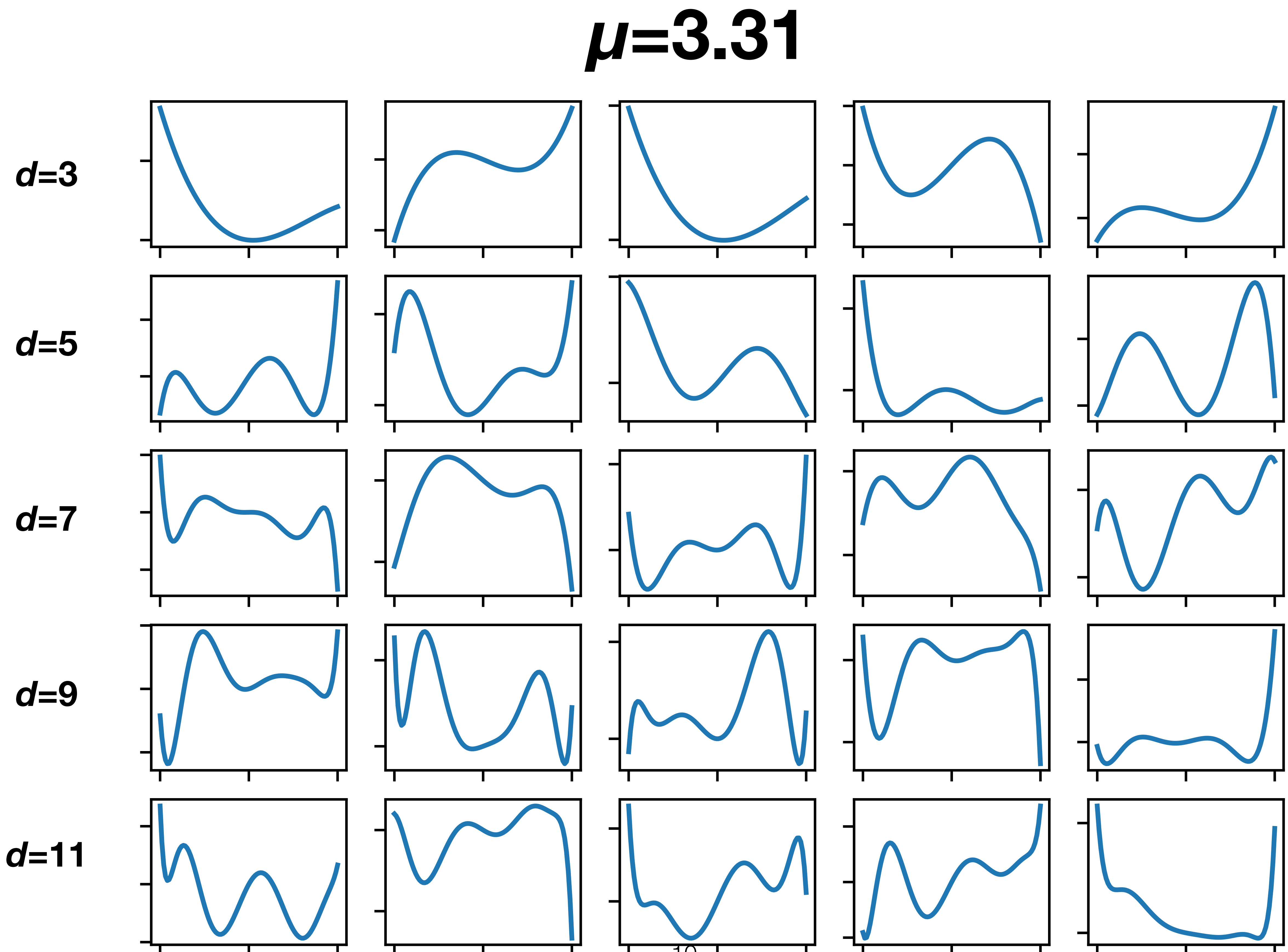


Random polynomials

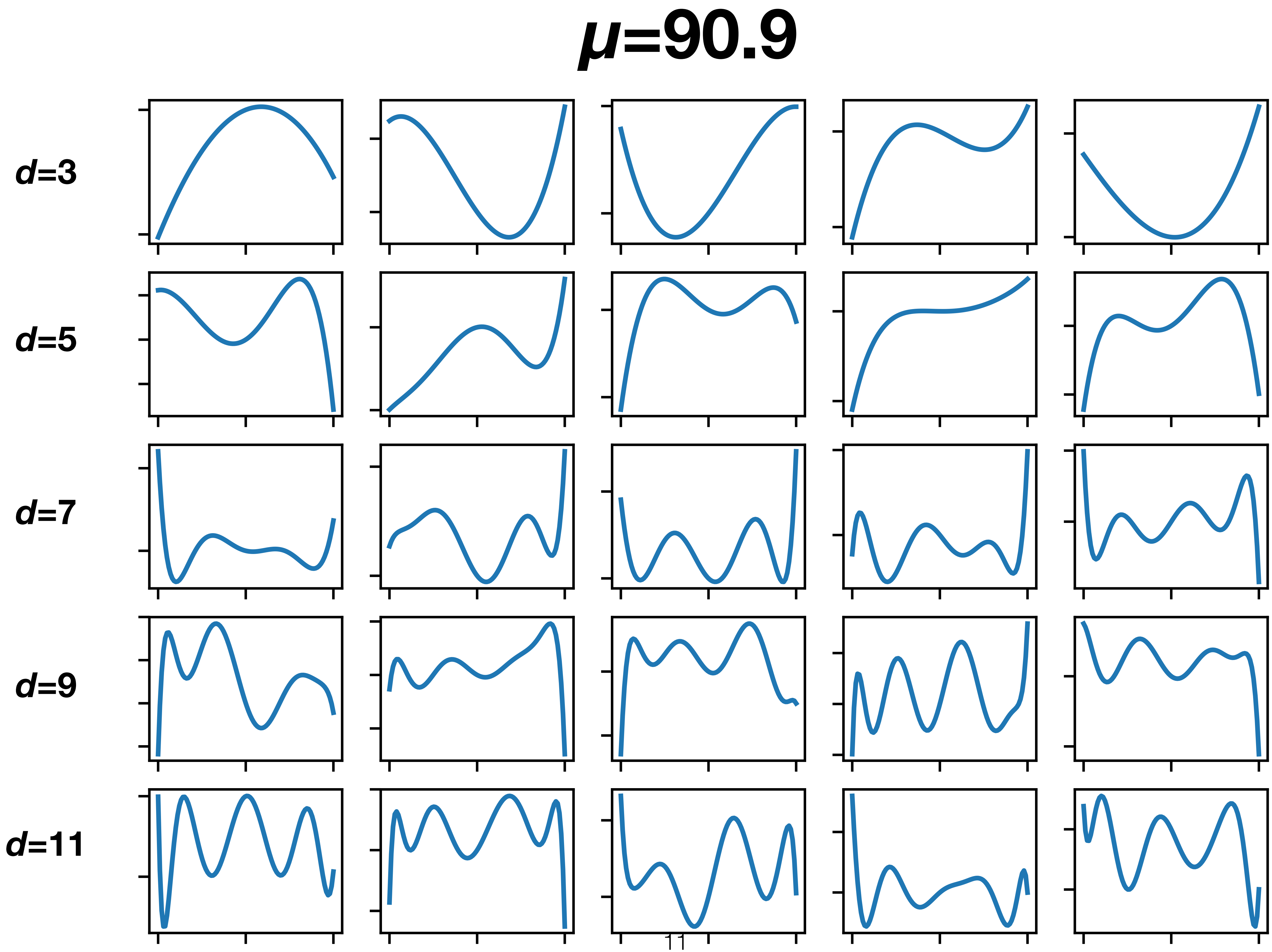
μ=0.058



Random polynomials

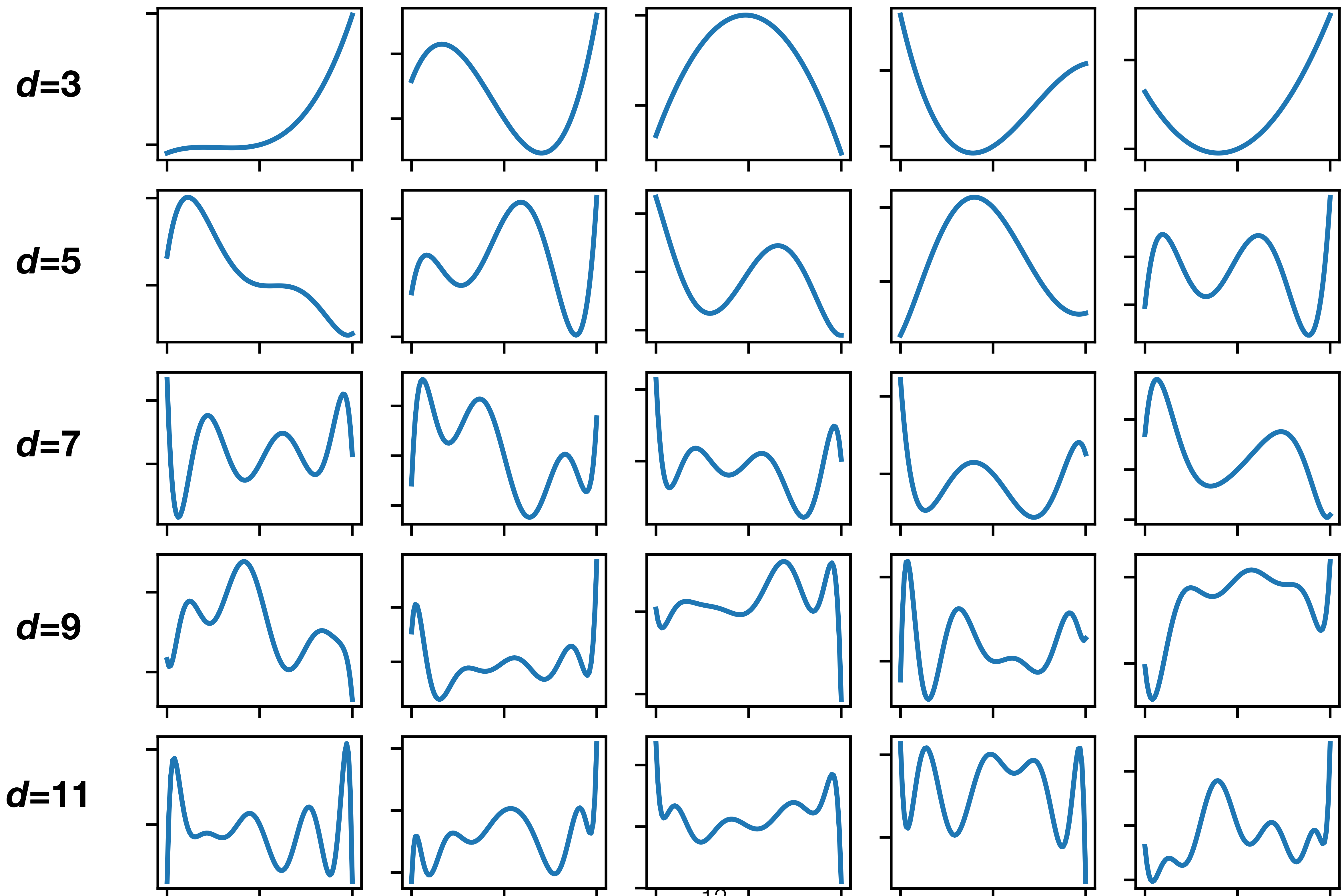


Random polynomials



Random polynomials

μ=537.8



Regularization

- The larger the coefficients (weights) **w** are allowed to be, the more the polynomial regressor can overfit.
- If we “encourage” the weights to be small, we can reduce overfitting.
- This is a form of **regularization** — any practice designed to improve the machine’s ability to **generalize** to new data.

Regularization

- One of the simplest and oldest regularization techniques is to *penalize* large weights in the cost function.

- We can define an extra cost:

$$\sum_{i=1}^{m} w_i^2 = \mathbf{w}^\top \mathbf{w}$$

- This is called a ***L*$_2$ regularization term**.

Regularization

- The ***L*₂-regularized** *f*_{MSE} becomes:

$$f_{\mathrm{MSE}}(\mathbf{w}) = \frac{1}{2n}(\hat{\mathbf{y}} - \mathbf{y})^\top(\hat{\mathbf{y}} - \mathbf{y}) + \frac{\alpha}{2n}\mathbf{w}^\top\mathbf{w}$$

- Here, α (typically a scalar) is the **regularization strength**.

Model complexity

Model complexity

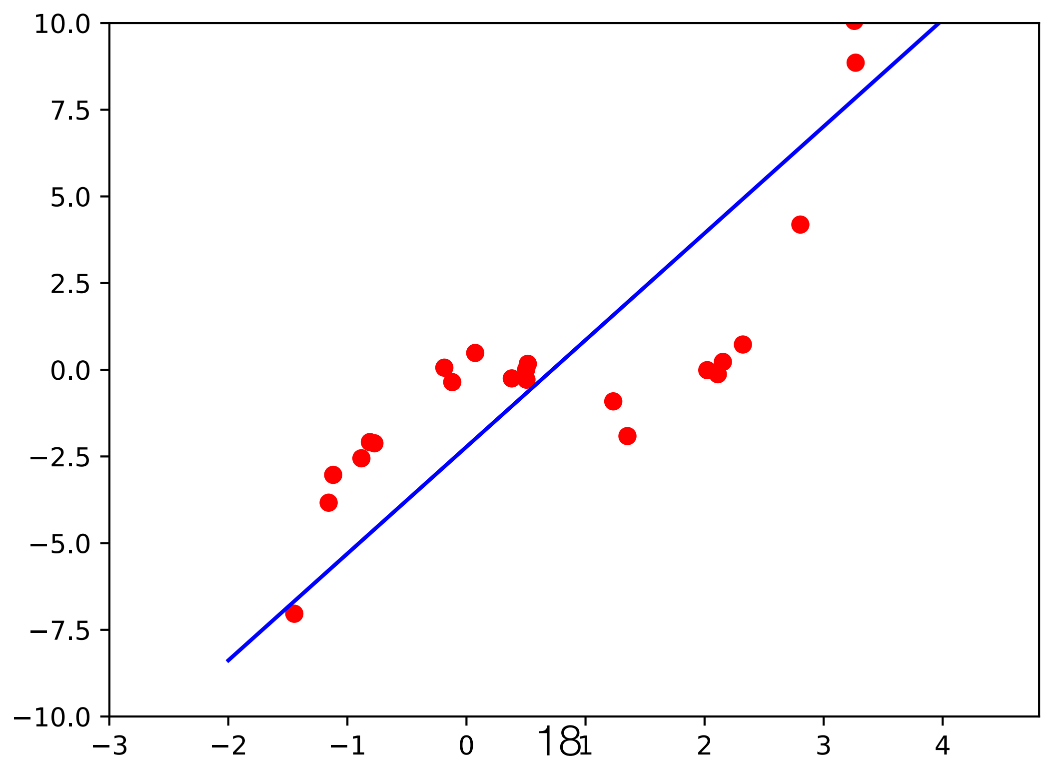
- With polynomial regression, we saw an easy way to increase the complexity of our ML model.
  - With higher degree *d*, our model becomes strictly more powerful.
  - With larger coefficients, the regression line becomes more flexible.

Model complexity

- ML models can fail (i.e., exhibit poor accuracy) due to two reasons:

1. **Bias**: the model is too simple to fit the data distribution ==> underfitting.

    - This can result in low accuracy during both training and testing.



underfitting

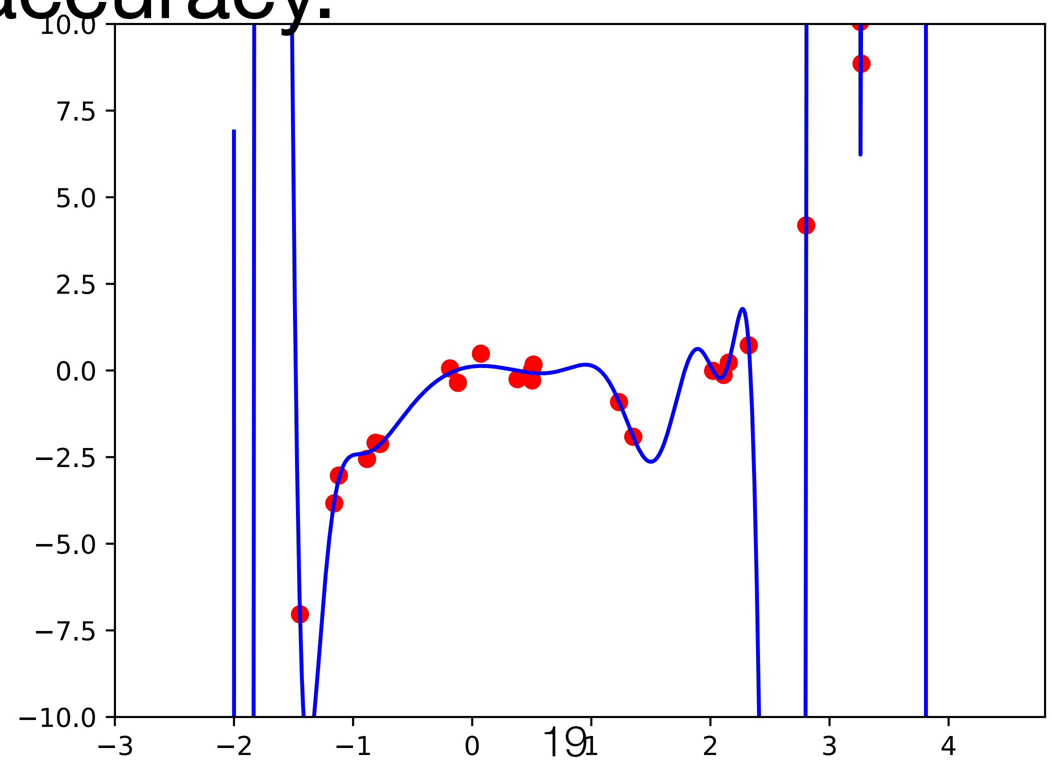
Jacob Whitehill, WPI

Model complexity

- ML models can fail (i.e., exhibit poor accuracy) due to two reasons:

  2. **Variance**: the model is too complex and is prone to overfitting. Re-training on different datasets will result in very different weight values.

     - This can result in high training accuracy but low testing accuracy.



overfitting

Jacob Whitehill, WPI

Model complexity

- Note that the degree of polynomial regression is just one kind of model complexity.
- Others:
  - Size of the input (24x24? 36x36?) to the machine.
  - Number of layers in a neural network (more later).

Model complexity

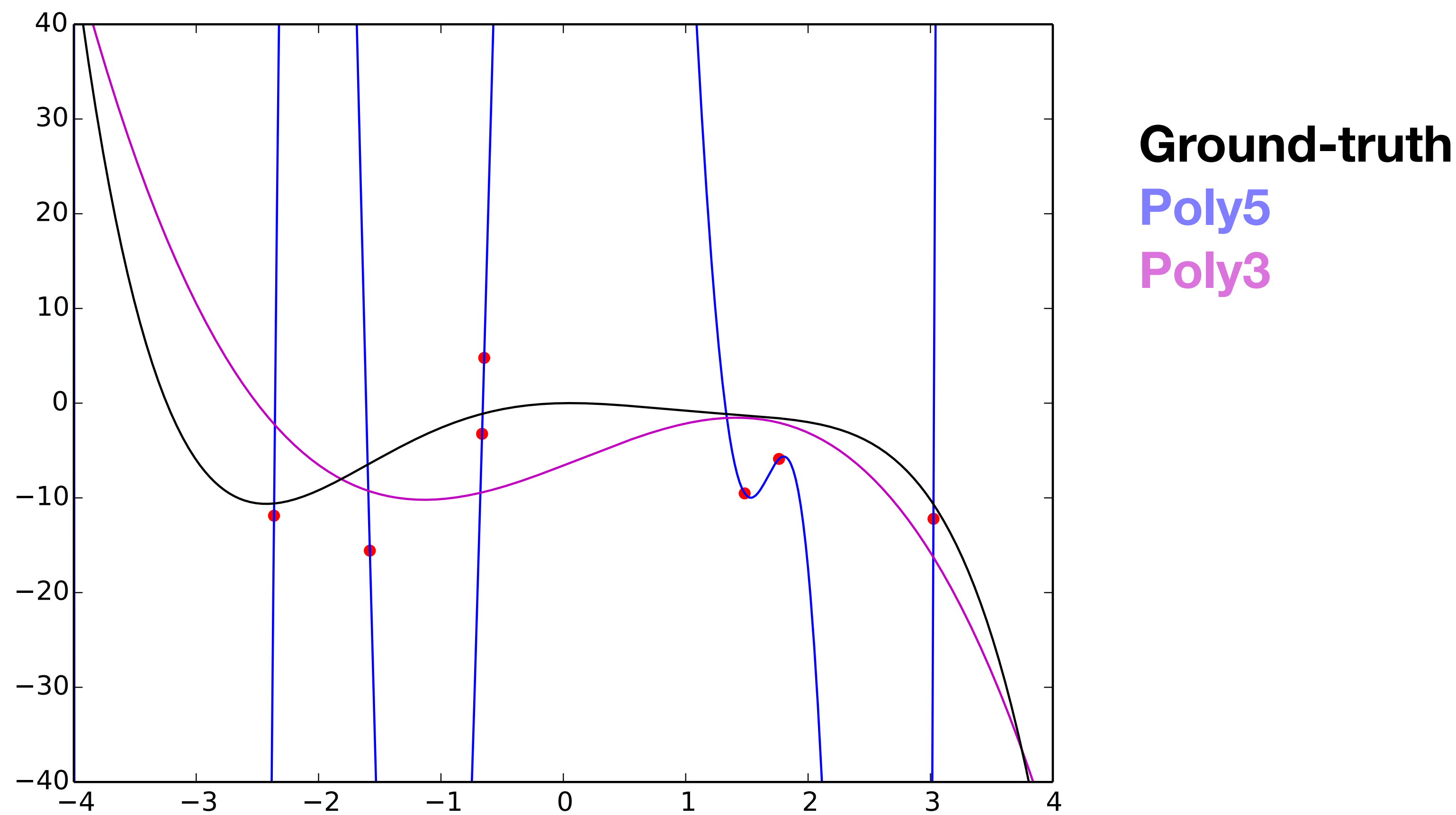
- In general: the more training data you have…
  - …the higher will be the testing accuracy of your trained machine.
  - …the more complex model you can use without overfitting.
  - … the less you need to regularize.

Model complexity

- In general: the more training data you have…
  - …the higher will be the testing accuracy of your trained machine.
  - …the more complex model you can use without overfitting.
  - … the less you need to regularize.
- Therefore, as your training dataset grows, you might decide to switch to a more powerful architecture.

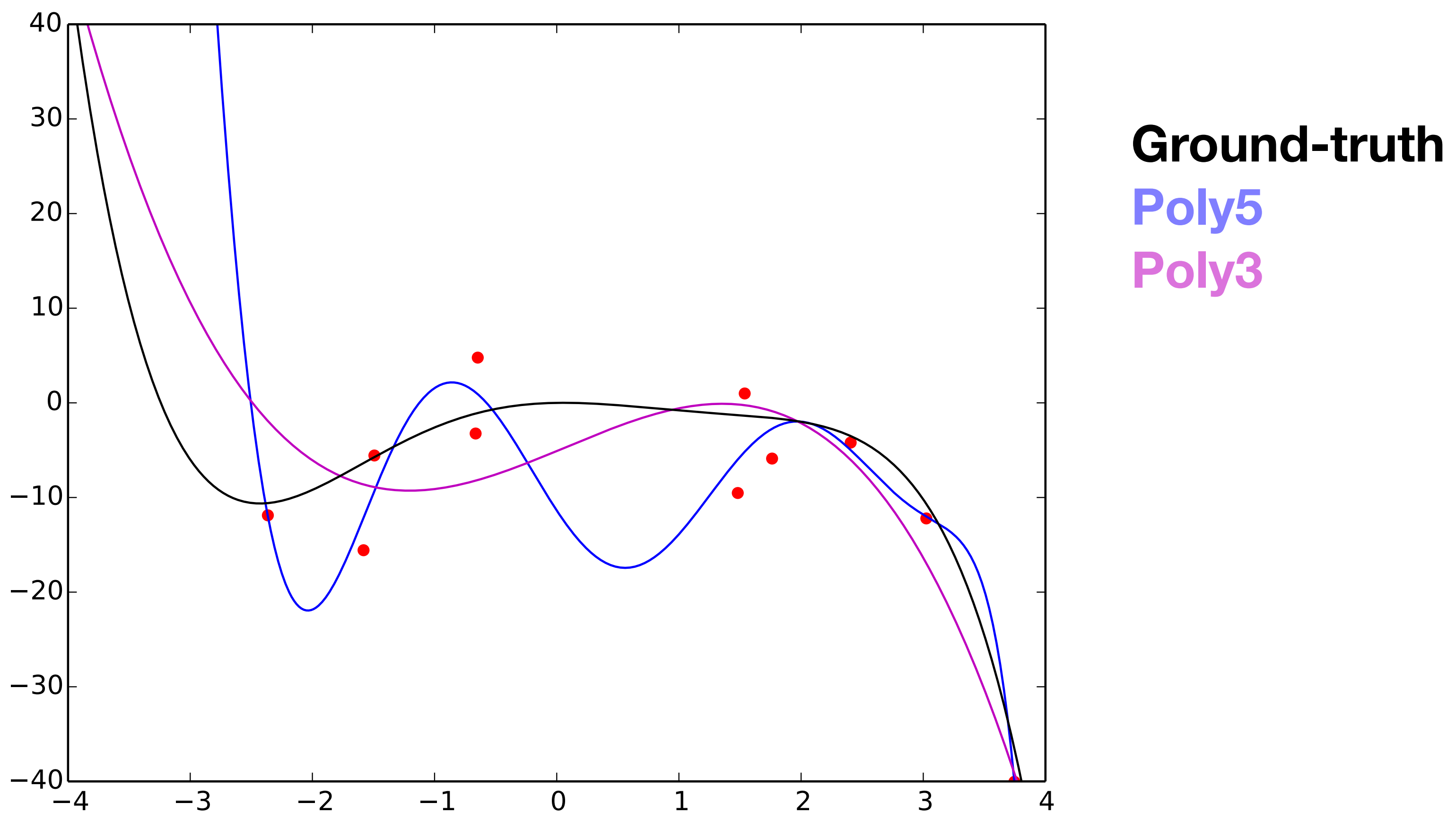
Illustration

- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x$ + noise
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.



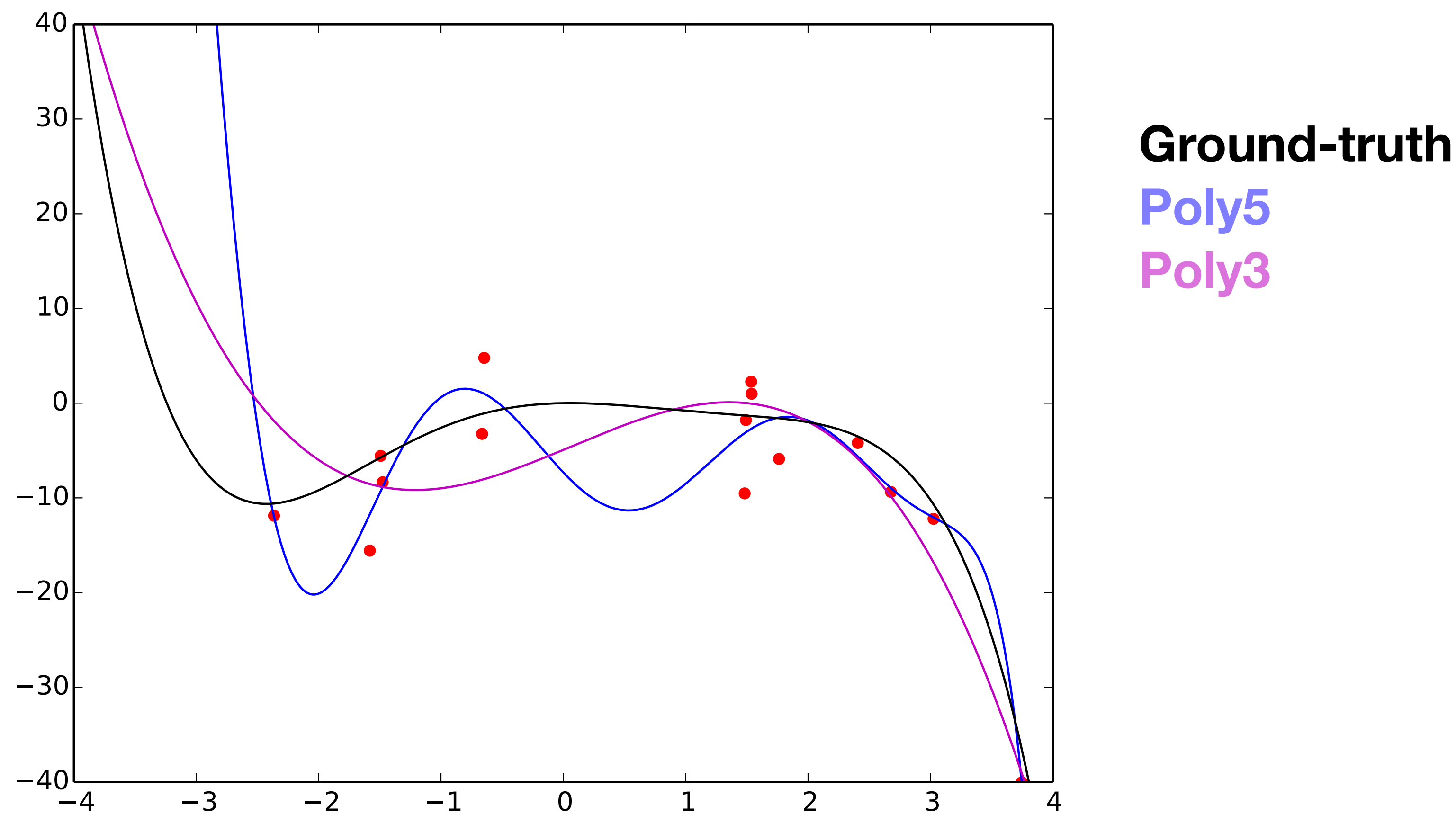
Illustration

- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x$ + noise
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.



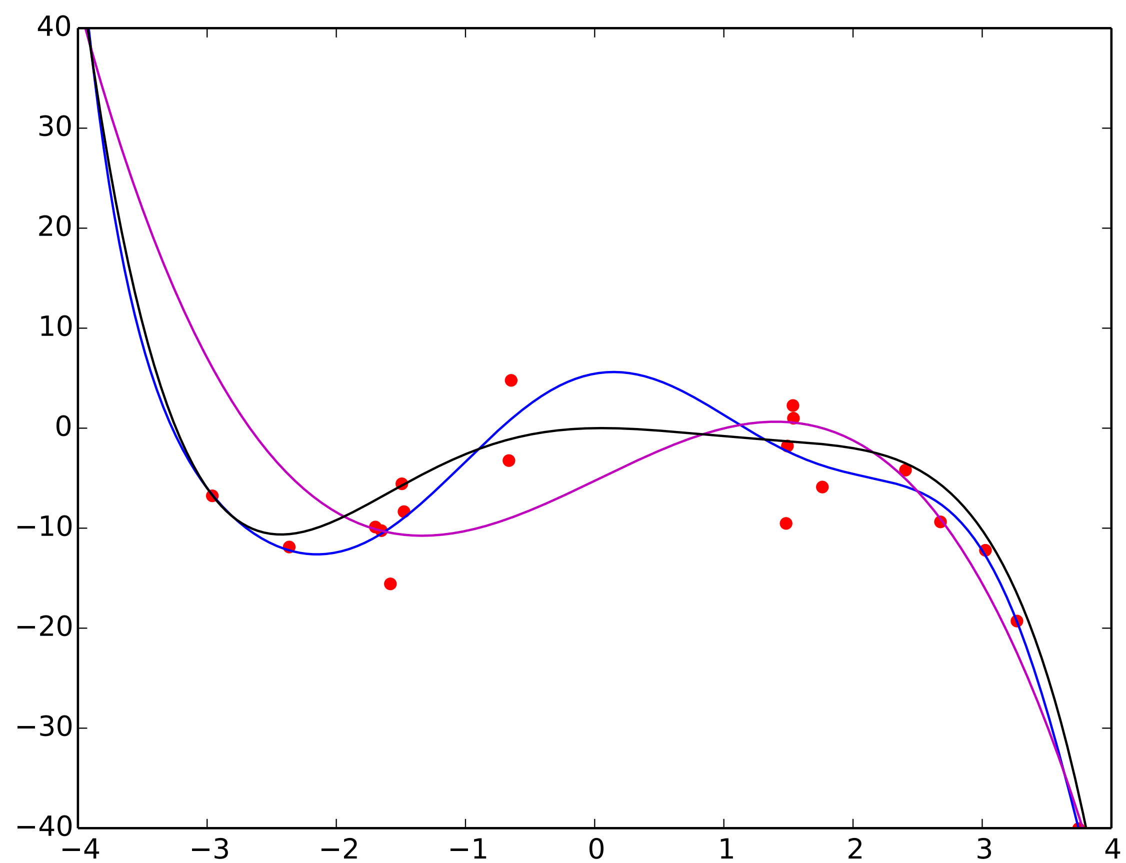
Illustration

- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x$ + noise
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.



Illustration

- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x$ + noise
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.

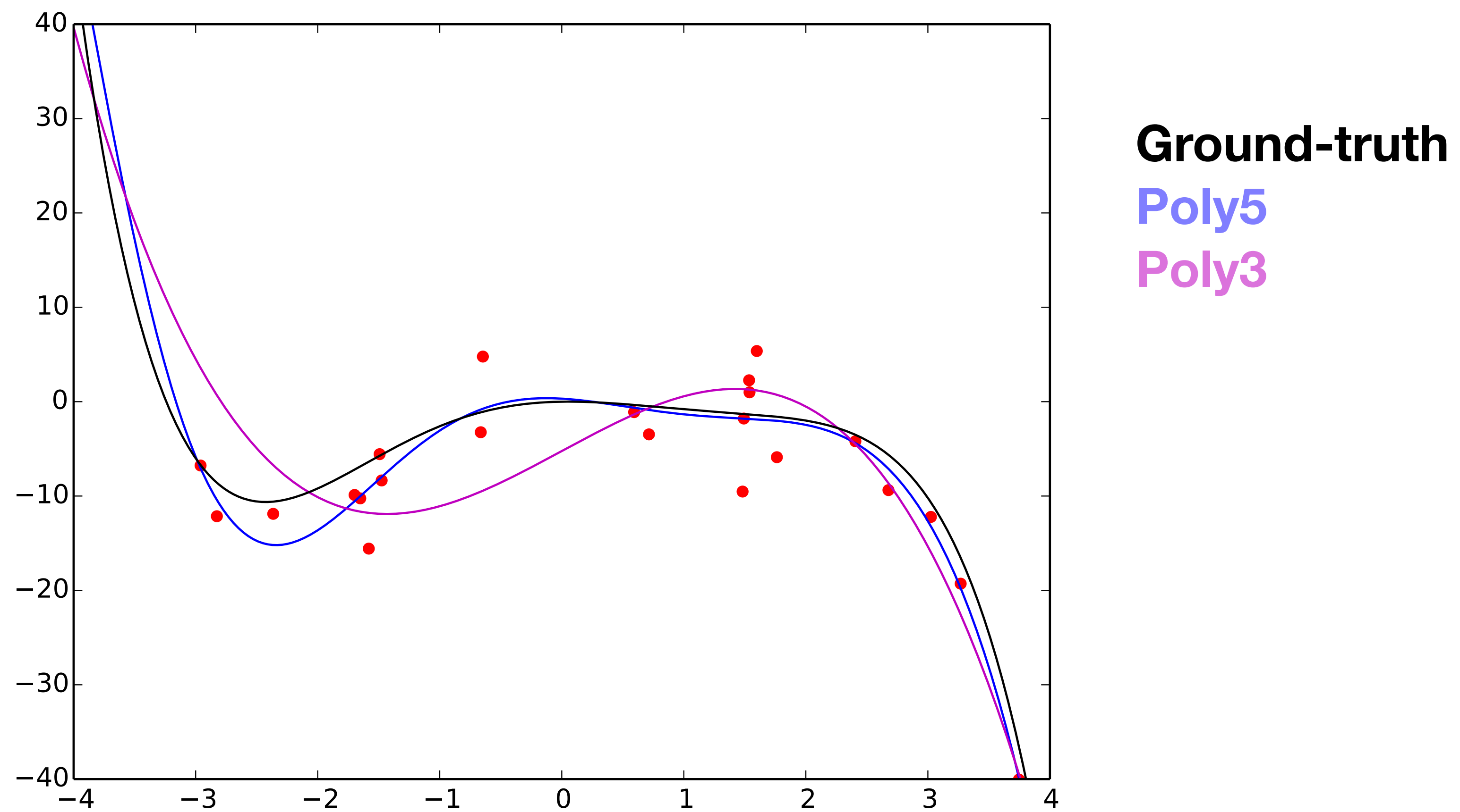


Ground-truth
Poly5
Poly3

Jacob Whitehill, WPI

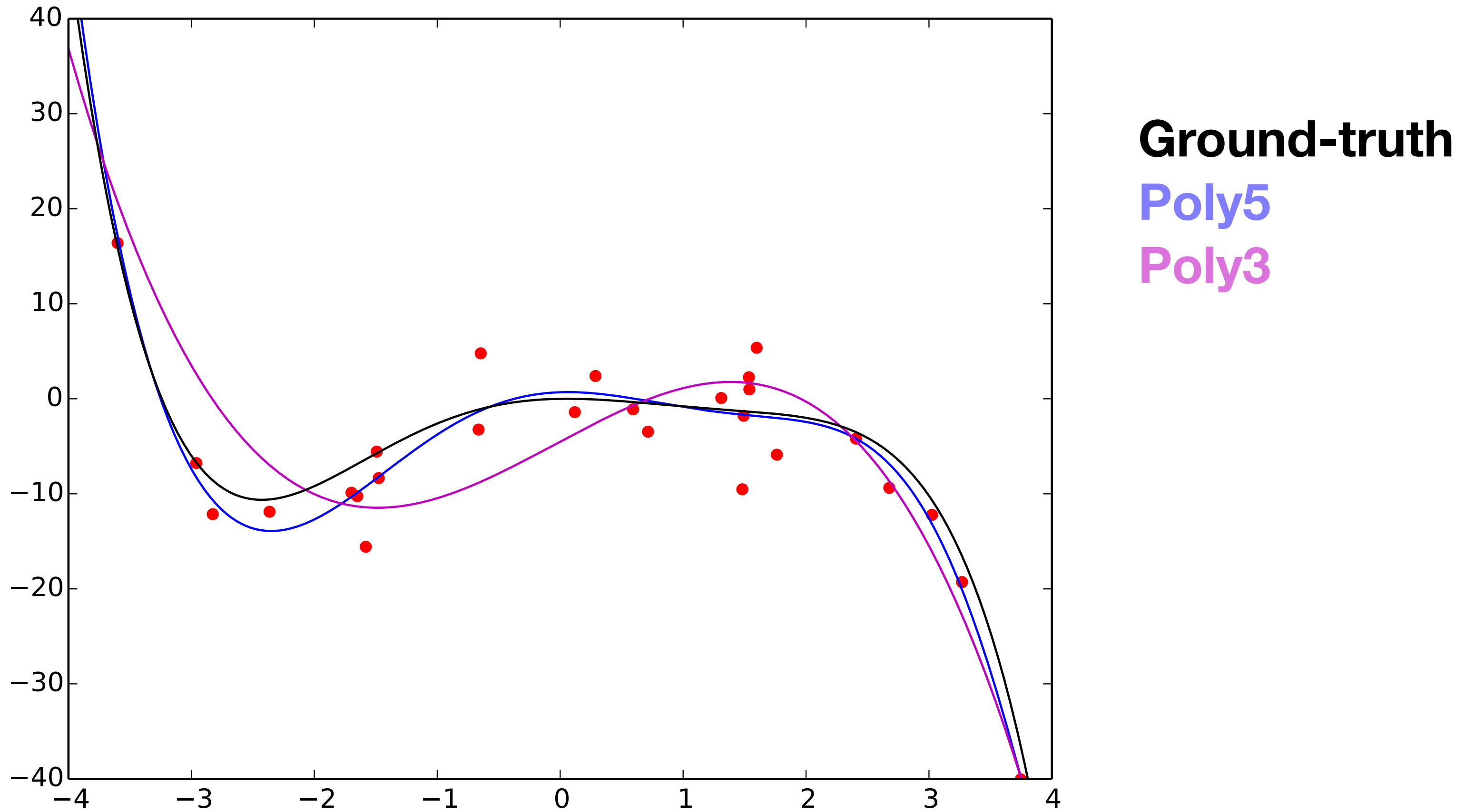
Illustration

- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x + \text{noise}$
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.



Illustration

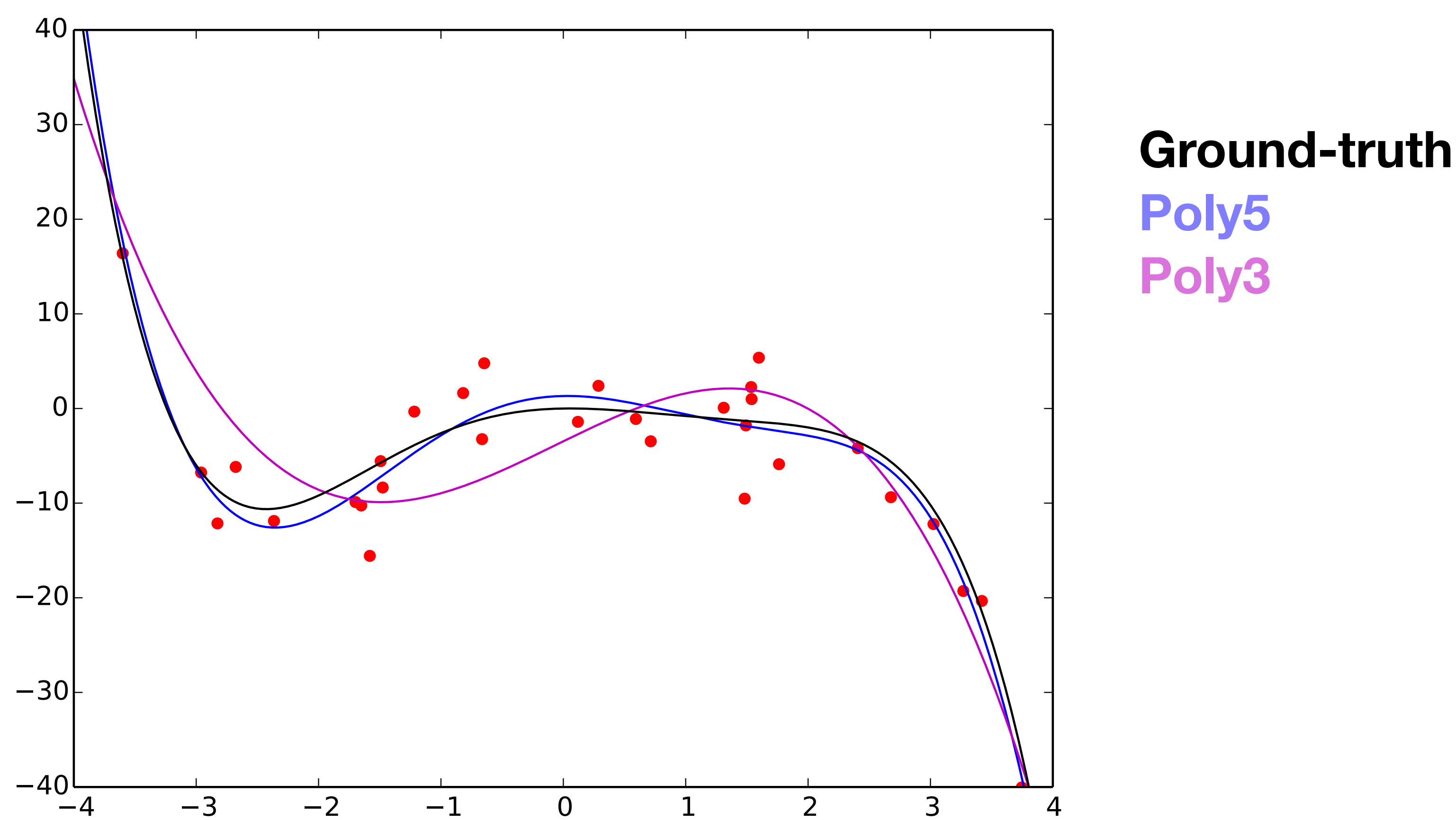
- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x + \text{noise}$
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.



Jacob Whitehill, WPI

Illustration

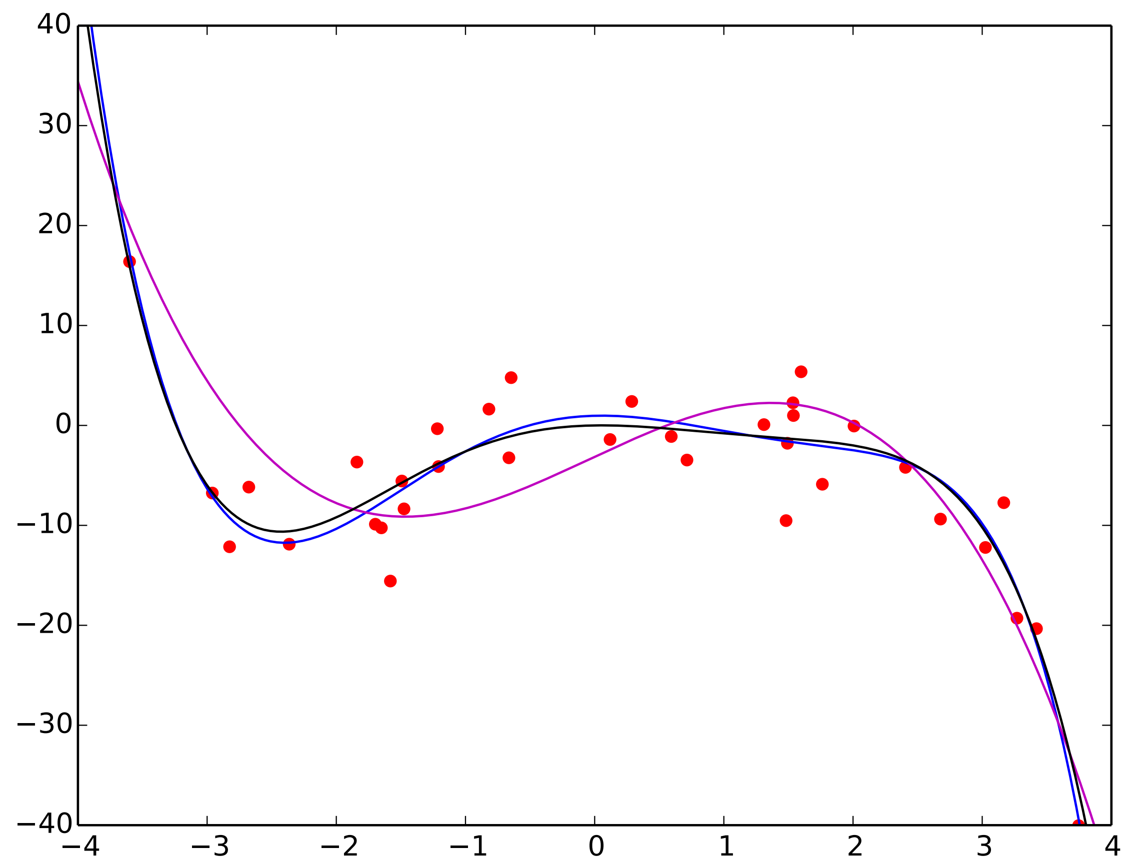
- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x$ + noise
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.



Jacob Whitehill, WPI

Illustration

- Simulation:
  - Ground-truth: $y = -0.1x^5 + 0.1x^4 + 0.8x^3 - 1.8x^2 + 0.2x$ + noise
  - At each round, we add 4 more data points.
  - We compare polynomial regressors of degree 3 and 5.



Ground-truth
Poly5
Poly3

By this point, the poly5 regressor is better than the poly3 regressor.

Regression for categorical data

Kaggle

- A handy resource to practice your ML skills is Kaggle, which is a website that hosts may machine learning competitions (with fabulous prizes).

- Example for House Prices: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

Case study: housing price prediction

- Suppose we want to predict housing prices, i.e., how much a house will sell for given a set of attributes (area, access to street, # fireplaces, etc.) about it.

- We could define a linear regression model:

  SalePrice = w_1 * Area + w_2 * NumFireplaces + … + b

- We can then use linear regression to train the optimal weights **w** to minimize the MSE.

Categorical variables

- Sometimes we might want to predict something (e.g., SalePrice) based on a variable that is not a number, e.g.:
  - LandContour of a house:
    - Level
    - Banked
    - Hillside
    - Depression

Categorical variables

- We can't multiply a number by a string!
- Two chief ways of handling this:
  - For ordinal relationships (e.g., Fair, Good, Very Good, Excellent), we can convert to a single integer variable.
  - For categorical relationships (e.g., Banked, Hillside, Depression), we can convert to binary **dummy variables** (aka **1-hot encoding**).

Categorical variables

- For every example in our dataset (both training and testing), we convert the single LandContour (LC) category variable into *K* binary dummy variables, where *K* is the number of possible categories, e.g.:
  - ```
    SalePrice,Area,LC
    100000,250,Hillside
    120520,280,Banked
    90500,220,Level
    110100,250,Banked
    …
    ```
    ```

Categorical variables

- For every example in our dataset (both training and testing), we convert the single LandContour (LC) category variable into *K* binary dummy variables, where *K* is the number of possible categories, e.g.:
    - ```
      SalePrice,Area,LC_Hill,LC_Bnk,LC_Lvl
      100000,250,1,0,0
      120520,280,0,1,0
      90500,220,0,0,1
      110100,250,0,1,0
      …
      ```
- For each training/testing example, the dummy variable LC_*Z* equals 1 if the LandContour has value *Z*, and 0 otherwise.
      ```

Categorical variables

- We can now define a linear regression using the dummy variables:

  SalePrice = w_1 * Area + w_2 * LC_Hill + w_3 * LC_Bnk +
  w_4 * LC_Lvl + … + b

- Weight w_2 specifies how much a Hillside land contour tends to increase the sale price of the house.

Model visualization

Model visualization

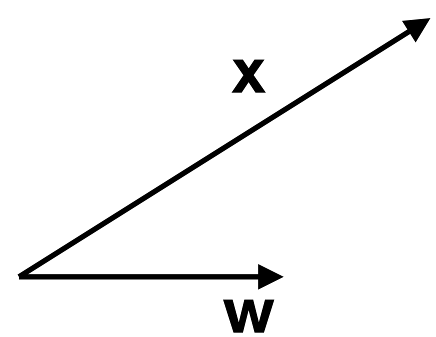
- It can be useful for debugging and verification purposes to examine what your model has learned.
  - E.g., you might find that your model has captured a spurious relationship between inputs and outputs.
  - Alternatively, your inspection might reveal that your data was fundamentally formatted incorrectly.
- One way of doing so is to visualize the learned parameters (i.e., weights) of your model.

Trained weights

- With linear regression, the machine computes the inner product between **x** and **w**, i.e., $\hat{y} = \mathbf{x}^\top \mathbf{w}$.

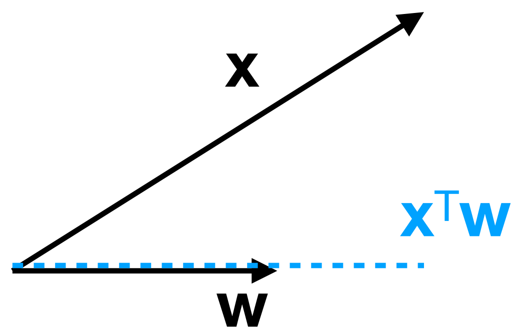
Trained weights

- With linear regression, the machine computes the inner product between $\mathbf{x}$ and $\mathbf{w}$, i.e., $\hat{y} = \mathbf{x}^\top\mathbf{w}$.
- The inner product $\mathbf{x}^\top\mathbf{w}$ computes the length of $\mathbf{x}$ *along direction* $\mathbf{w}$.



Trained weights

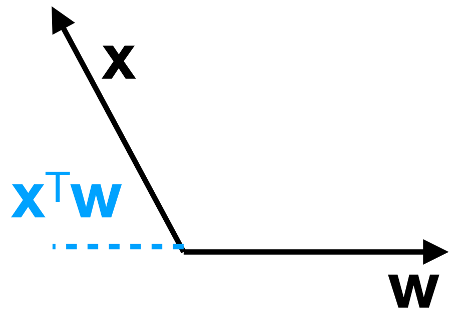
- With linear regression, the machine computes the inner product between $\mathbf{x}$ and $\mathbf{w}$, i.e., $\hat{y} = \mathbf{x}^\top \mathbf{w}$.
- The inner product $\mathbf{x}^\top \mathbf{w}$ computes the length of $\mathbf{x}$ *along direction* $\mathbf{w}$.



Here, $\mathbf{x}^\top \mathbf{w} > 0$ since they point (partly) in the same direction.

Trained weights

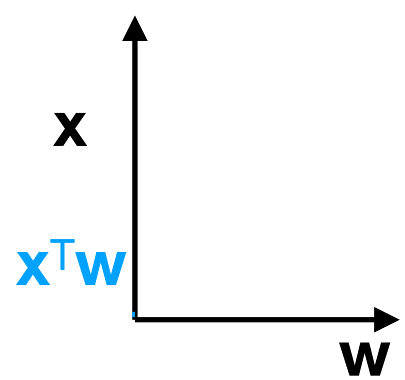
- With linear regression, the machine computes the inner product between **x** and **w**, i.e., $\hat{y} = \mathbf{x}^\top \mathbf{w}$.
- The inner product $\mathbf{x}^\top \mathbf{w}$ computes the length of **x** *along direction* **w**.



Here, $\mathbf{x}^\top \mathbf{w} < 0$ since they point (partly) in opposite directions.

Trained weights

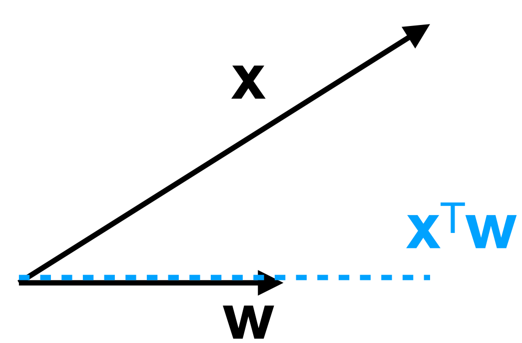
- With linear regression, the machine computes the inner product between **x** and **w**, i.e., $\hat{y} = \mathbf{x}^\top \mathbf{w}$.
- The inner product $\mathbf{x}^\top \mathbf{w}$ computes the length of **x** *along direction* **w**.



Here, $\mathbf{x}^\top \mathbf{w} = 0$ since they x and w are orthogonal.

Trained weights

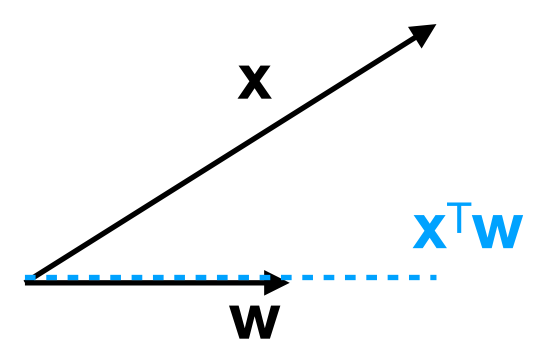
- With linear regression, the machine computes the inner product between $\mathbf{x}$ and $\mathbf{w}$, i.e., $\hat{y} = \mathbf{x}^\top \mathbf{w}$.
- The inner product $\mathbf{x}^\top \mathbf{w}$ computes the length of $\mathbf{x}$ *along direction* $\mathbf{w}$.



- In this sense, the inner product measures the "compatibility" between $\mathbf{x}$ and $\mathbf{w}$.

Trained weights

- With linear regression, the machine computes the inner product between **x** and **w**, i.e., $\hat{y} = \mathbf{x}^\top \mathbf{w}$.
- The inner product $\mathbf{x}^\top \mathbf{w}$ computes the length of **x** *along direction* **w**.



- In this sense, the inner product measures the “compatibility” between **x** and **w**.

Demo with linear regression for age regression.

Logistic regression

Using linear regression for classification

- In homework 1, you used step-wise classification to train a smile classifier.
- In homework 2, you will use linear regression to train an age estimator.
- Can we use linear regression to train a smile classifier?

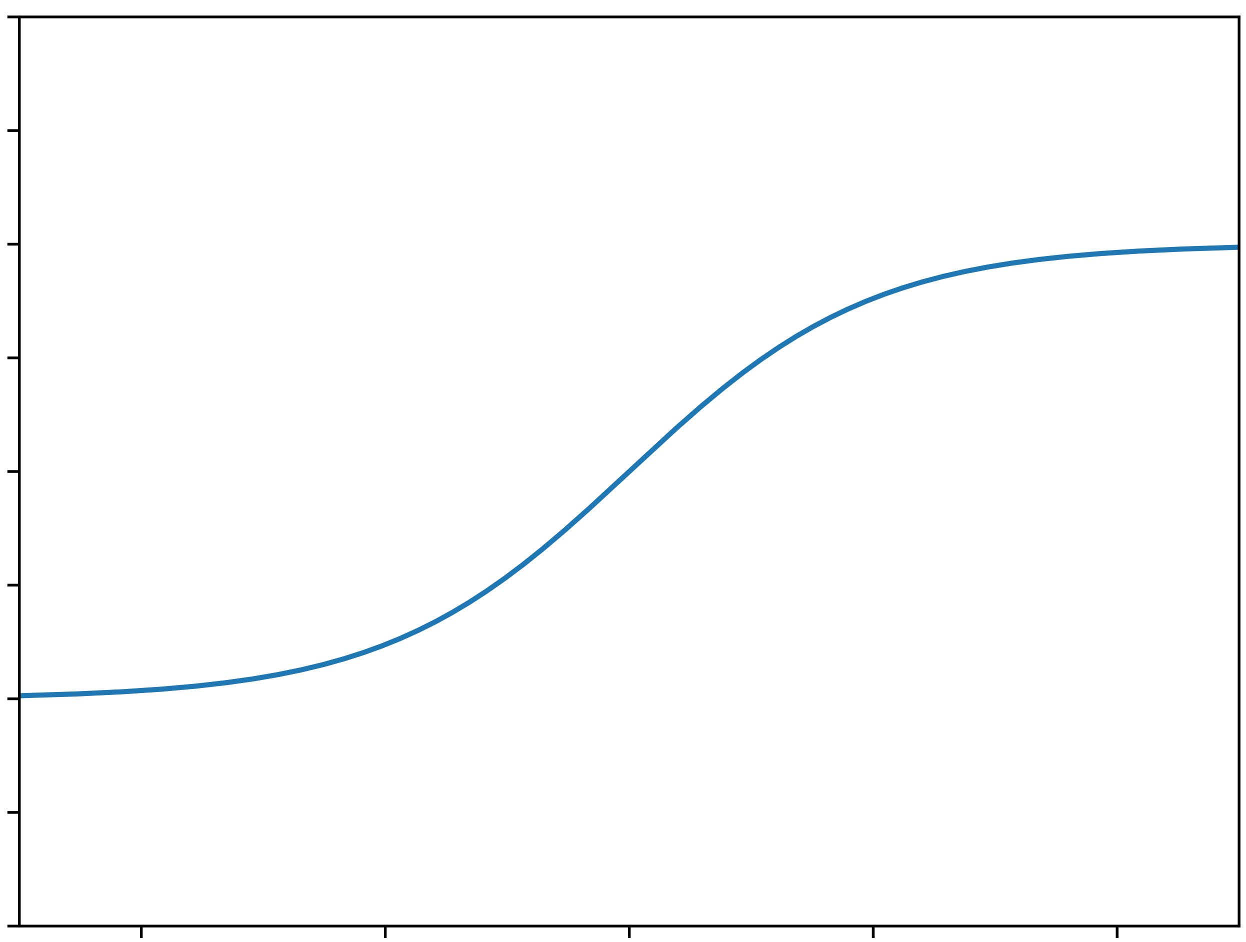
Using linear regression for classification

- While in principle we could, it is somewhat unnatural.
- Recall the distinction between:
  - **Regression**: predict any real number.
  - **Classification**: choose from a finite set (e.g., {0, 1}).
- Since a linear regression machine can output *any real number*, then any guess $\hat{y} < 0$ or $\hat{y} > 1$ is always a mistake!
- Why not instead “squash” the output to always lie in (0,1)?

Sigmoid: a “squashing” function

- A sigmoid function is an “s”-shaped, monotonically increasing and bounded function.

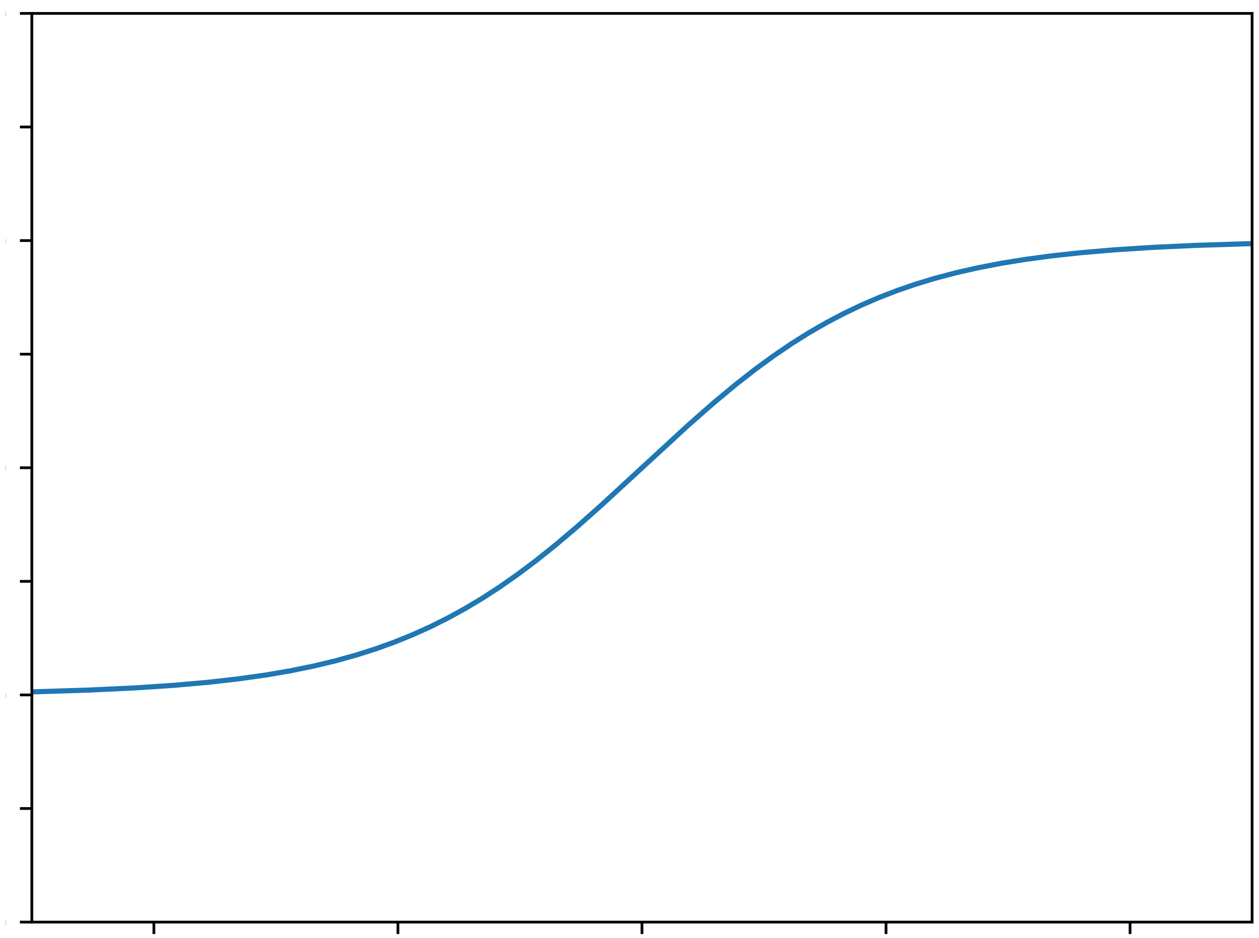
- Here is an example of a sigmoid function:



Sigmoid: a "squashing" function

- A sigmoid function is an "s"-shaped, monotonically increasing and bounded function.

- Here is an example of a sigmoid function:



Which function(s) could describe this curve?

1. $$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

2. $$\frac{e^x + e^{-x}}{e^x - e^{-x}}$$

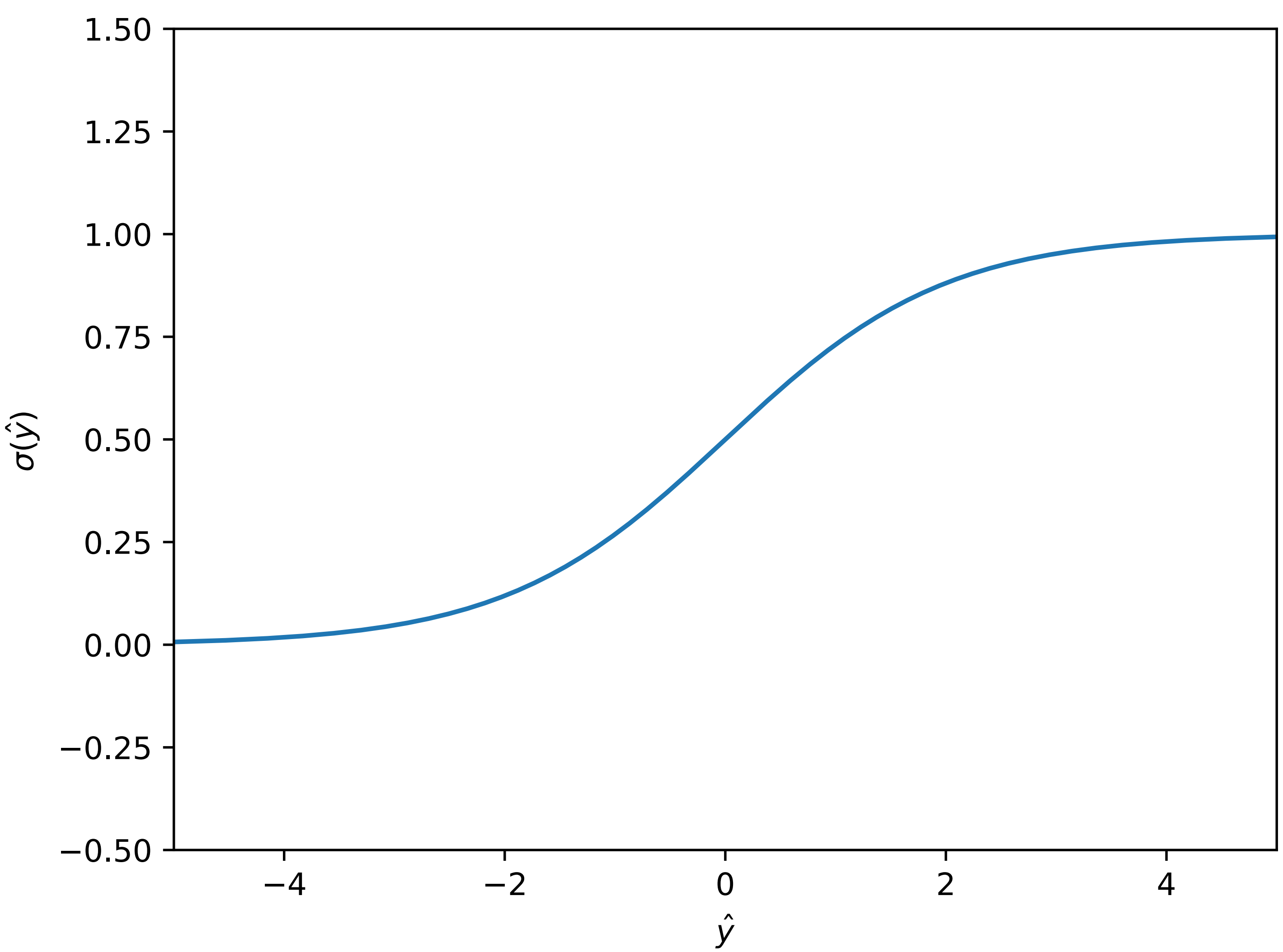
3. $$\frac{1}{1 - e^{-x}}$$

4. $$\frac{1}{1 + e^{-x}}$$

Sigmoid: a “squashing” function

- A sigmoid function is an “s”-shaped, monotonically increasing and bounded function.

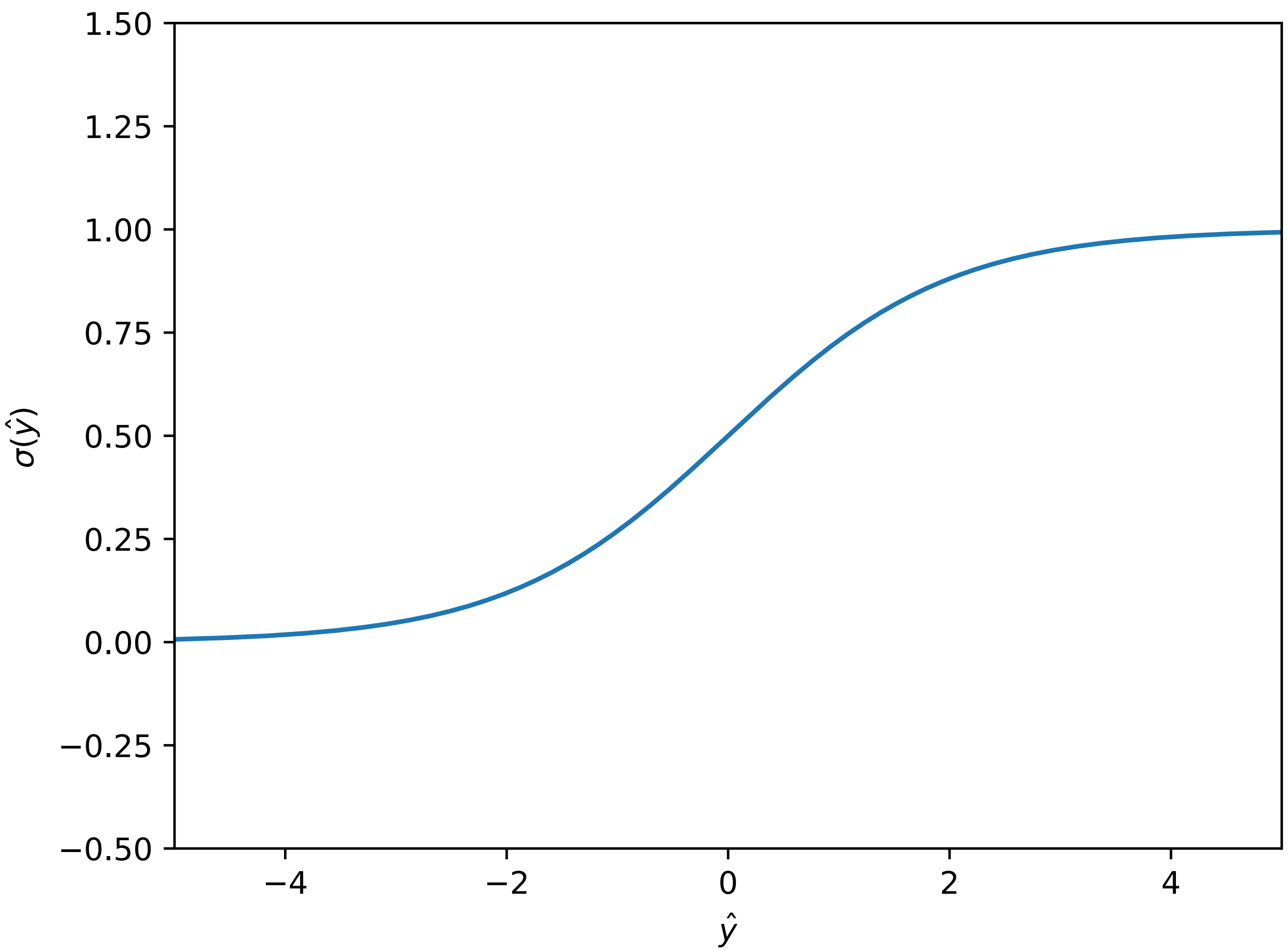
- Here is the **logistic sigmoid** function σ:



Sigmoid: a “squashing” function

- A sigmoid function is an “s”-shaped, monotonically increasing and bounded function.

- Here is the **logistic sigmoid** function σ:



Which function(s) describe(s) the curve?

1. $$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

2. $$\frac{e^x + e^{-x}}{e^x - e^{-x}}$$

3. $$\frac{1}{1 - e^{-x}}$$

4. $$\frac{1}{1 + e^{-x}}$$

Logistic sigmoid

- The logistic sigmoid function σ has some nice properties:
  - σ(-*z*) = 1 - σ(*z*)

$$\begin{aligned}
\sigma(z) &= \frac{1}{1+e^{-z}} \\
1-\sigma(z) &= 1-\frac{1}{1+e^{-z}} \\
&= \frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}} \\
&= \frac{e^{-z}}{1+e^{-z}} \\
&= \frac{1}{1/e^{-z}+1} \\
&= \frac{1}{1+e^{z}} \\
&= \sigma(-z)
\end{aligned}$$

Logistic sigmoid

- The logistic sigmoid function σ has some nice properties:
  - $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

$$
\begin{aligned}
\sigma(z) &= \frac{1}{1+e^{-z}} \\
\frac{\partial \sigma}{\partial z} = \sigma'(z) &= -\frac{1}{(1+e^{-z})^2}(e^{-z} \times (-1)) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{e^{-z}}{1+e^{-z}} \times \frac{1}{1+e^{-z}} \\
&= \frac{1}{1/e^{-z}+1} \times \frac{1}{1+e^{-z}} \\
&= \frac{1}{1+e^{z}} \times \frac{1}{1+e^{-z}} \\
&= \sigma(z)(1-\sigma(z))
\end{aligned}
$$

Logistic regression

- With logistic regression, our predictions are defined as:

$$\hat{y} = \sigma\left(\mathbf{x}^\top \mathbf{w}\right)$$

- Hence, they are forced to be in (0,1).
- For classification, we can interpret the real-valued outputs as probabilities that express how confident we are in a prediction, e.g.:
    - $\hat{y}$=0.95: very confident that the class is a smile.
    - $\hat{y}$=0.45: not very confident that the class is a non-smile.

Logistic regression

- How to train? Unlike linear regression, logistic regression has no analytical (“one-shot”) solution.
  - We can use gradient descent instead.
  - We have to apply the **chain-rule of differentiation** to handle the sigmoid function.

Gradient descent for *linear* regression

- First, recall the gradient of the f_{MSE} for *linear* regression.

- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}(\mathbf{x}^\top \mathbf{w} - y)^2 \\
&= \mathbf{x}(\mathbf{x}^\top \mathbf{w} - y) \\
&= \mathbf{x}(\hat{y} - y)
\end{aligned}
$$

Gradient descent for *linear* regression

- First, recall the gradient of the f_{MSE} for *linear* regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}(\mathbf{x}^\top \mathbf{w} - y)^2 \\
&= \mathbf{x}(\mathbf{x}^\top \mathbf{w} - y) \\
&= \mathbf{x}(\hat{y} - y)
\end{aligned}
$$

How far the prediction is from ground-truth.

Gradient descent for *linear* regression

- First, recall the gradient of the f_{MSE} for *linear* regression.

- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}(\mathbf{x}^\top \mathbf{w} - y)^2 \\
&= \mathbf{x}(\mathbf{x}^\top \mathbf{w} - y) \\
&= \mathbf{x}(\hat{y} - y)
\end{aligned}
$$

How the prediction *ŷ* depends on the weights *w*.

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$f_{\text{MSE}}(\mathbf{w}) \quad = \quad \frac{1}{2}(\hat{y} - y)^2$$

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.

- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\mathrm{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2
\end{aligned}
$$

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2 \\
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2\right]
\end{aligned}
$$

$= ?$

Recall:
$\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Attenuated gradient

- What if the weights **w** are initially chosen badly, so that $\hat{y}$ is very close to 1, even though y = 0 (or vice-versa)?
  - Then $\hat{y}(1 - \hat{y})$ is close to 0.
- In this case, the gradient:

$$\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) = \mathbf{x}\left(\hat{y} - y\right)\hat{y}\left(1 - \hat{y}\right)$$

  will be very close to 0.

- If the gradient is 0, then no learning will occur!

Different cost function

- For this reason, logistic regression is typically trained using a different cost function from f_{MSE}.

- One particularly well-suited cost function uses logarithms.

- Logarithms and the logistic sigmoid interact well:

$$
\begin{aligned}
\frac{\partial}{\partial z}\left[\log \sigma(z)\right] &= \frac{1}{\sigma(z)}\sigma'(z) \\
&= \frac{1}{\sigma(z)}\sigma(z)(1-\sigma(z)) \\
&= 1-\sigma(z)
\end{aligned}
$$

The gradient of log(σ) is surprisingly simple.

Different cost function

- For this reason, logistic regression is typically trained using a different cost function from f_{MSE}.

- One particularly well-suited cost function uses logarithms.

- Logarithms and the logistic sigmoid interact well:

$$\frac{\partial}{\partial \mathbf{w}} \left[\log \sigma(\mathbf{x}^\top \mathbf{w}) \right] \quad =$$

Different cost function

- For this reason, logistic regression is typically trained using a different cost function from f_{MSE}.
- One particularly well-suited cost function uses logarithms.
- Logarithms and the logistic sigmoid interact well:

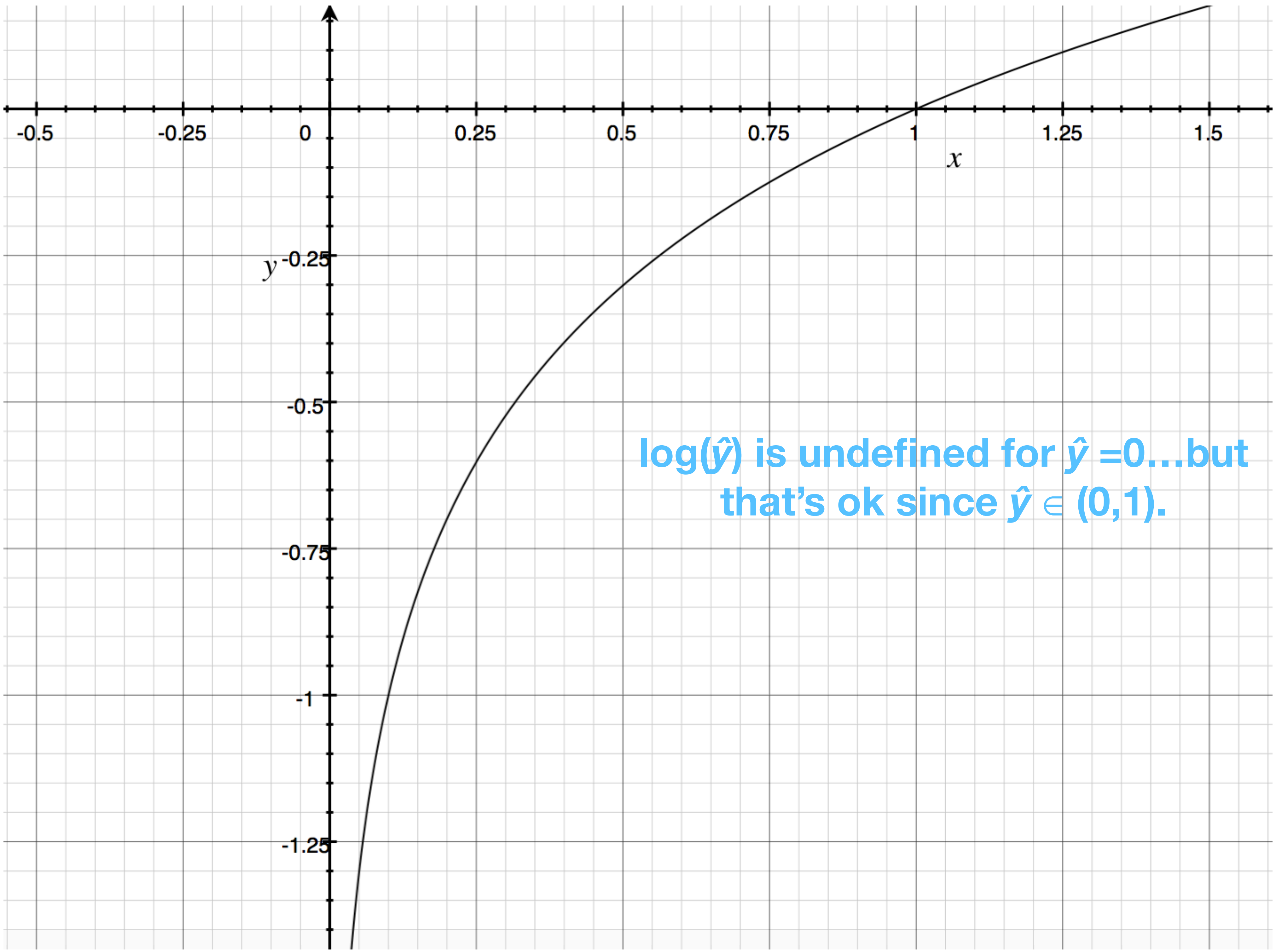
$$\frac{\partial}{\partial \mathbf{w}}\left[\log \sigma(\mathbf{x}^\top \mathbf{w})\right] = \frac{1}{\sigma(\mathbf{x}^\top \mathbf{w})}\sigma(\mathbf{x}^\top \mathbf{w})\left(1 - \sigma(\mathbf{x}^\top \mathbf{w})\right)\mathbf{x}$$

Different cost function

- For this reason, logistic regression is typically trained using a different cost function from f_{MSE}.
- One particularly well-suited cost function uses logarithms.
- Logarithms and the logistic sigmoid interact well:

$$
\begin{aligned}
\frac{\partial}{\partial \mathbf{w}} \left[\log \sigma(\mathbf{x}^\top \mathbf{w}) \right] &= \frac{1}{\sigma(\mathbf{x}^\top \mathbf{w})} \sigma(\mathbf{x}^\top \mathbf{w}) \left(1 - \sigma(\mathbf{x}^\top \mathbf{w}) \right) \mathbf{x} \\
&= \left(1 - \sigma(\mathbf{x}^\top \mathbf{w}) \right) \mathbf{x}
\end{aligned}
$$

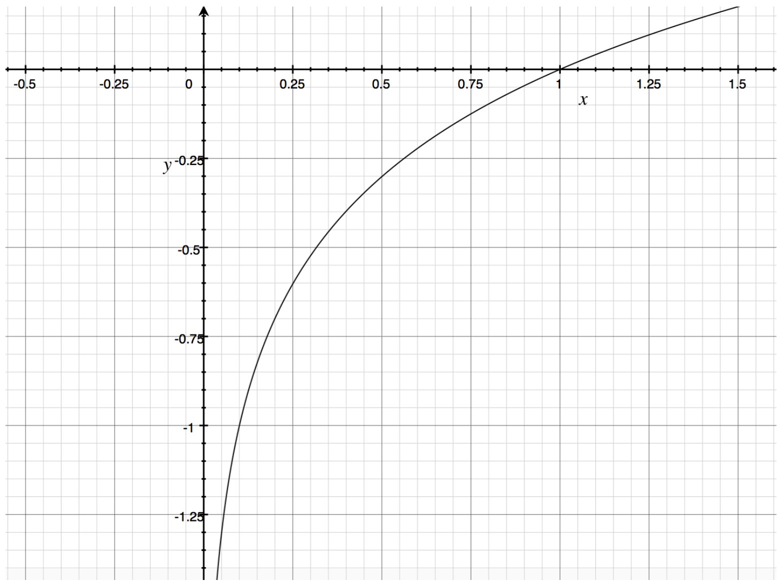
Logarithm function



Log loss

- How could we define a "log-loss" function $f_{\log}$ so that:
  - $f_{\log}(y, \hat{y})$ is small when $\hat{y} \approx y$ and large when they are far apart.

1. $-y \log \hat{y} - \hat{y} \log y$
2. $-y \log \hat{y} - (1-y) \log \hat{y}$
3. $-y \log \hat{y} - (1-y) \log(1-\hat{y})$
4. $-(1-y) \log \hat{y} - y \log(1-\hat{y})$



log($\hat{y}$)

Gradient descent for logistic regression with log-loss

$$\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) \quad = \quad \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \right) \right]$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1 - \sigma(\mathbf{x}^\top \mathbf{w})) \right)
\end{aligned}
$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[-\left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1 - \sigma(\mathbf{x}^\top \mathbf{w})) \right) \\
&= -\left(y \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{\sigma(\mathbf{x}^\top \mathbf{w})} \qquad\qquad\qquad\qquad \right)
\end{aligned}
$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1 - \sigma(\mathbf{x}^\top \mathbf{w})) \right) \\
&= -\left(y \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{\sigma(\mathbf{x}^\top \mathbf{w})} - (1-y) \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{1-\sigma(\mathbf{x}^\top \mathbf{w})} \right)
\end{aligned}
$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1 - \sigma(\mathbf{x}^\top \mathbf{w})) \right) \\
&= - \left(y \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{\sigma(\mathbf{x}^\top \mathbf{w})} - (1-y) \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{1-\sigma(\mathbf{x}^\top \mathbf{w})} \right) \\
&= - \left(y\mathbf{x}(1-\sigma(\mathbf{x}^\top \mathbf{w})) - (1-y)\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w}) \right)
\end{aligned}
$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1 - \sigma(\mathbf{x}^\top \mathbf{w})) \right) \\
&= -\left(y \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{\sigma(\mathbf{x}^\top \mathbf{w})} - (1-y) \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{1-\sigma(\mathbf{x}^\top \mathbf{w})} \right) \\
&= -\left(y\mathbf{x}(1-\sigma(\mathbf{x}^\top \mathbf{w})) - (1-y)\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w}) \right) \\
&= -\mathbf{x} \left(y - y\sigma(\mathbf{x}^\top \mathbf{w}) - \sigma(\mathbf{x}^\top \mathbf{w}) + y\sigma(\mathbf{x}^\top \mathbf{w}) \right) \\
&= -\mathbf{x} \left(y - \sigma(\mathbf{x}^\top \mathbf{w}) \right) \\
&= \mathbf{x}(\hat{y} - y)
\end{aligned}
$$

Same as for linear regression!

Linear regression versus logistic regression

	Linear regression	Logistic regression
Primary use	Regression	Classification
Prediction ($\hat{y}$)	$\hat{y} = \mathbf{x}^\top \mathbf{w}$	$\hat{y} = \sigma(\mathbf{x}^\top \mathbf{w})$
Loss	f_{MSE}	$f_{\log}$
Gradient	$\mathbf{x}(\hat{y} - y)$	$\mathbf{x}(\hat{y} - y)$

Linear regression versus logistic regression

	Linear regression	Logistic regression
Primary use	Regression	Classification
Prediction ($\hat{y}$)	$\hat{y} = \mathbf{x}^\top\mathbf{w}$	$\hat{y} = \sigma(\mathbf{x}^\top\mathbf{w})$
Loss	f_{MSE}	$f_{\log}$
Gradient	$\mathbf{x}(\hat{y} - y)$	$\mathbf{x}(\hat{y} - y)$

- Logistic regression is used primarily for *classification* even though it's called "regression".
- Logistic regression is an instance of a **generalized linear model** — a linear model combined with a **link function** (e.g., logistic sigmoid).
  - In neural networks, link functions are typically called **activation functions.**