Training and Overfitting

MMCi Block 2

Matthew Engelhard

Strategy: determine how small changes in parameters affect the loss

MORTALITY PREDICTION WORKSHEET

COVARIATES

patient	age	age_normalized	female	temp	temp_normalized
0	30.5	-0.5	0	105.0	2.4
1	74.0	1.1	1	96.7	-0.8
2	27.4	-0.6	0	96.1	-1.0
3	0.1	-1.5	1	98.5	-0.1
4	0.7	-1.5	1	96.5	-0.9
5	49.9	0.2	1	97.1	-0.6
6	72.9	1.0	1	100.1	0.5
7	29.1	-0.5	1	99.6	0.3
8	83.5	1.4	1	100.6	0.7
9	82.3	1.4	1	95.2	-1.3
10	23.7	-0.7	0	99.4	0.2
11	12.9	-1.1	0	96.6	-0.8
12	53.9	0.4	1	100.3	0.6
13	18.8	-0.9	0	98.6	0.0
14	51.8	0.3	0	98.5	-0.1
15	3.3	-1.4	0	94.6	-1.6
16	69.7	0.9	0	99.1	0.1
17	60.4	0.6	1	104.2	2.1
18	73.6	1.1	1	99.1	0.1
19	53.3	0.3	1	99.1	0.1

OUTCOMES AND PREDICTIONS

mortality	predicted_log_odds	predicted_prob	prediction	correct?	loss
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010
1	0.00	0.50	0	0	0.3010
1	0.00	0.50	0	0	0.3010
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
1	0.00	0.50	0	0	0.3010
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010

PARAMETERS

	b_age	b_female		b_temp	bias
guess	0.00	0.00		0.00	0.00
optimal					

PERFORMANCE

accuracy	avg_loss
0.65	0.3010

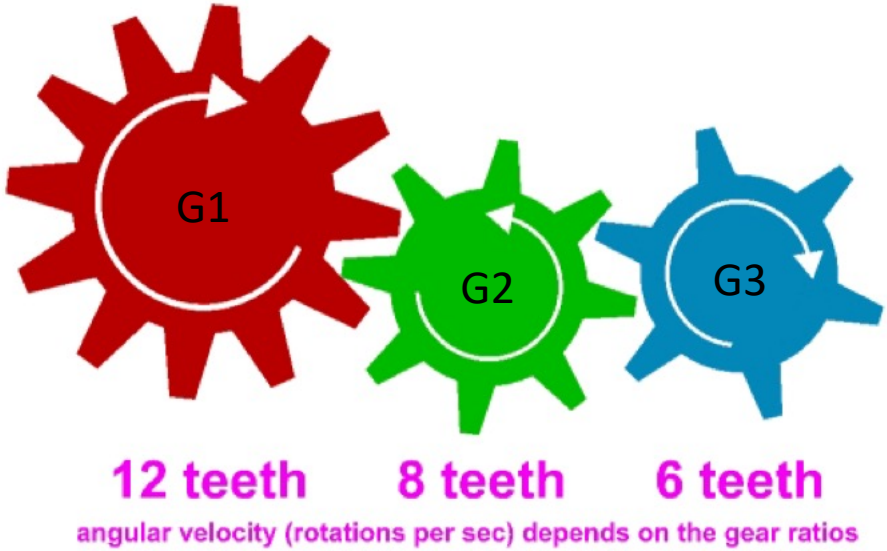
Strategy: determine how small changes in parameters affect the loss

We know:

- If we rotate G1 by 1 radian, G2 will rotate by –12/8 radians.
- If G2 rotates by 1 radian, G3 will rotate by -8/6 radians.

How do we determine the effect of G1 on G3?

- Multiply the effects.
- $\left(-\frac{12}{8}\right) * \left(-\frac{8}{6}\right) = \frac{12}{6} = 2$



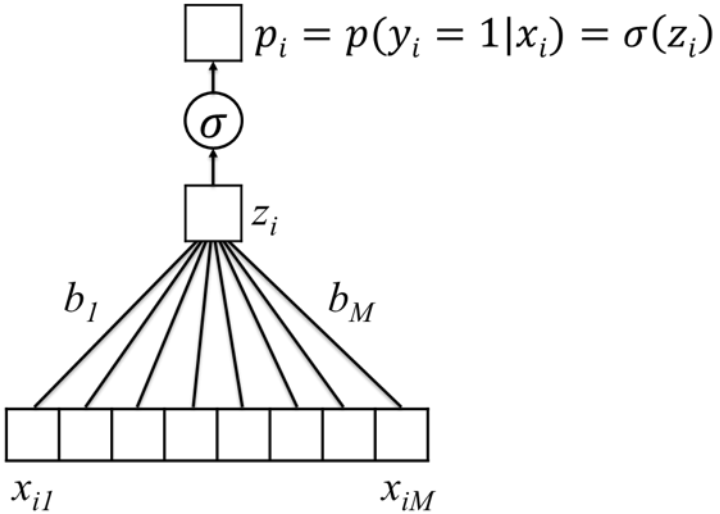
Strategy: determine how small changes in parameters affect the loss

We know:

- If we increase b_1 by a small amount ε, then z_i will increase by $\varepsilon * x_{i1}$
- If we increase z_i by a small amount ε, then p_i will increase by $\varepsilon * \frac{d\sigma(z_i)}{dz_i}$ (depends on z_i)

How do we determine the effect b_1 on the cross-entropy loss (which depends on p_i)?

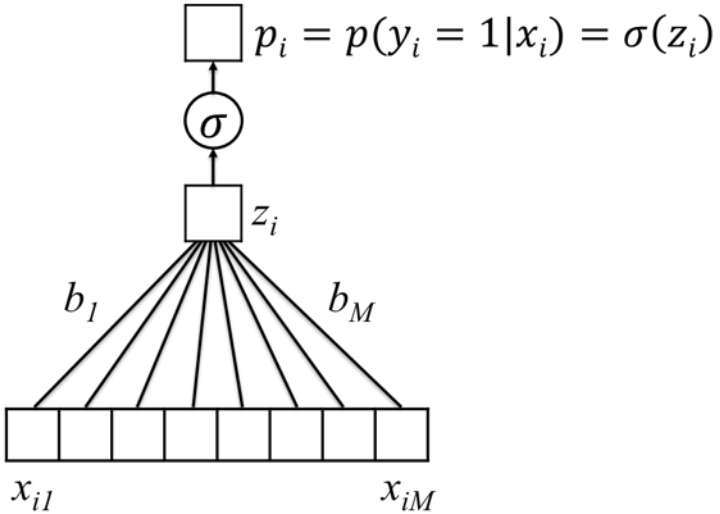
➢ Multiply the effects.



Strategy: determine how small changes in parameters affect the loss

This is called the *chain rule* (calc 101)

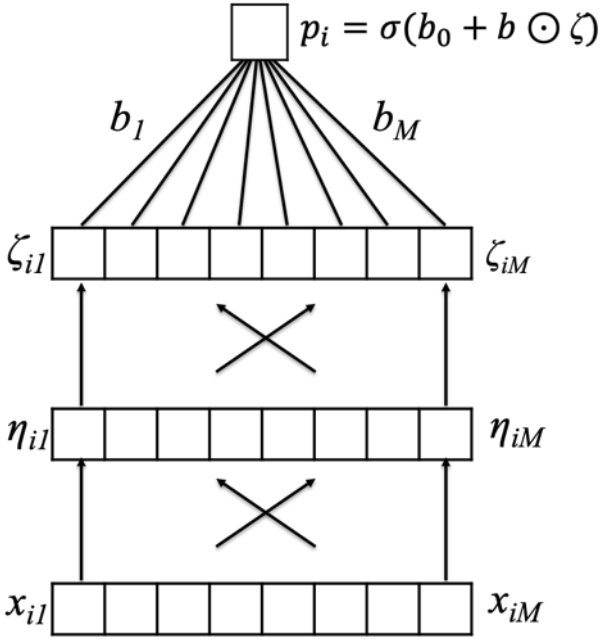
- We use it to see how small changes in parameters affect the loss
- Could be a very long chain…
- Some parameters have a greater effect than others
- We change all parameters at once, with each change proportional to that parameter's effect on the loss
    - This is *gradient descent*



Strategy: determine how small changes in parameters affect the loss

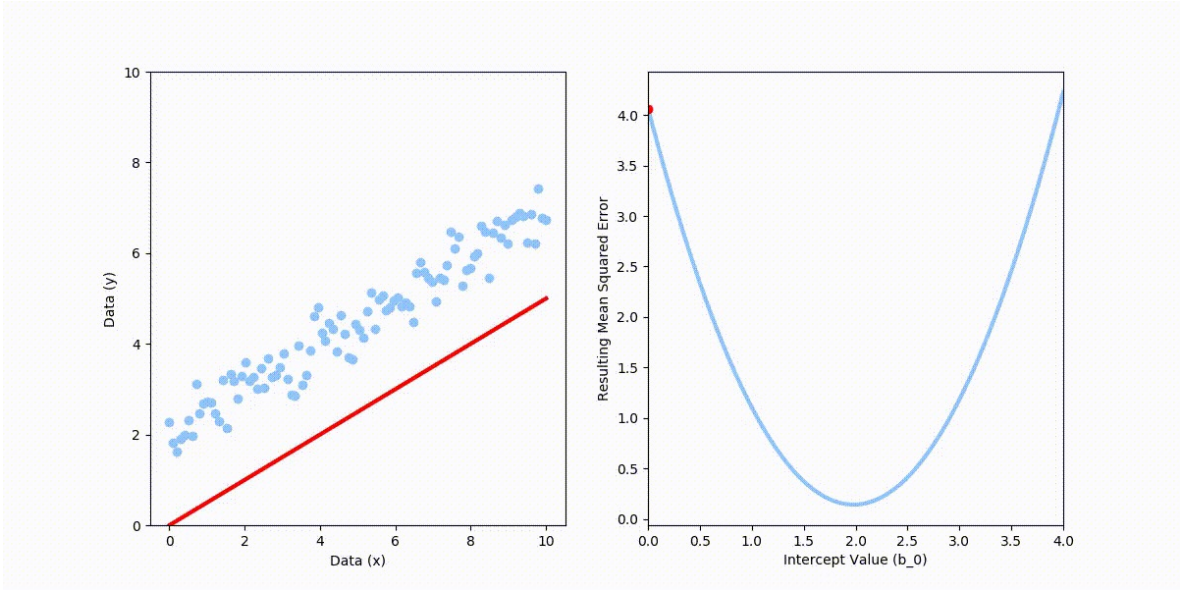
<u>It could be a very long (and complex) chain…</u>

- If we increase x_{i1} by ε, it changes *all* of the η_{ij}…
- …each of which changes *all* of the ζ_{ij}
- …each of which changes p_i
- Machine learning software like TensorFlow allows us to keep track, even for very complicated models



For simple models, minimizing the loss is easy.

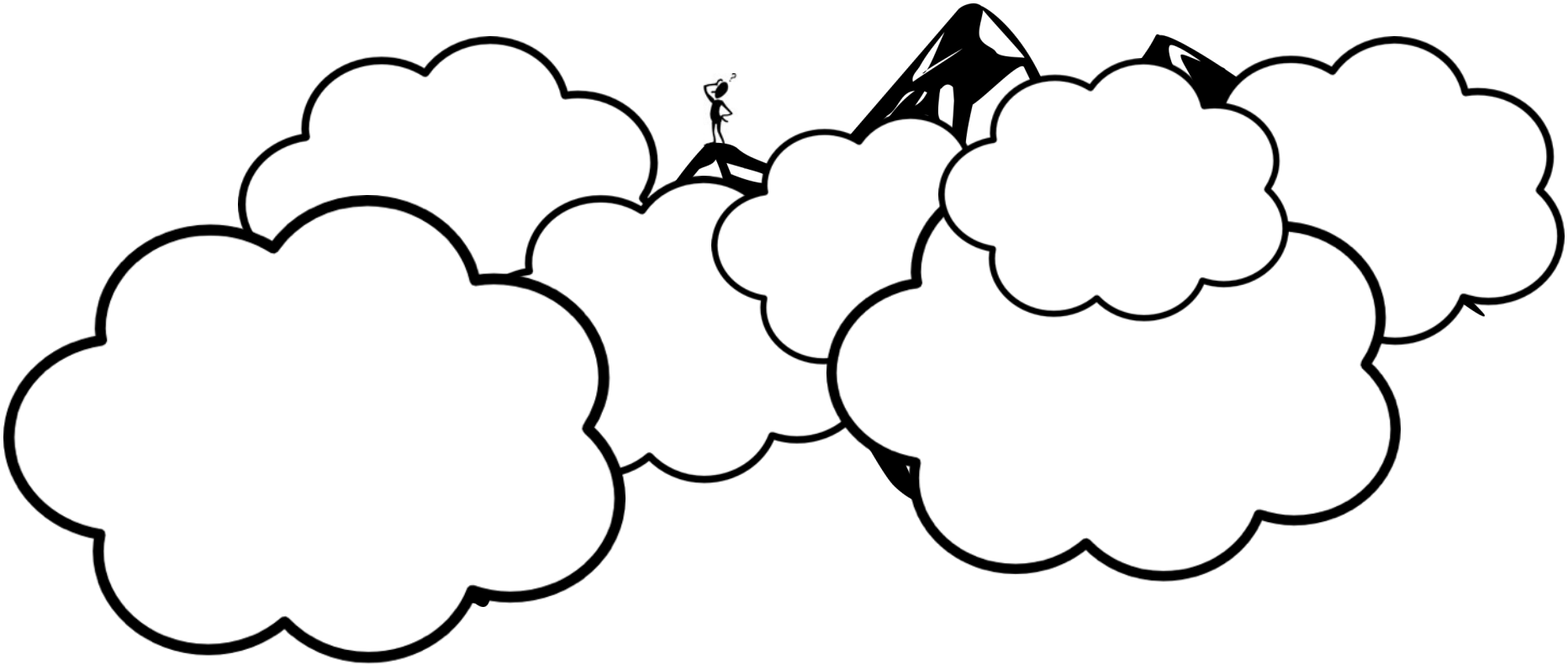
1. There are a limited number of parameters to consider
2. Here, the loss is ‘bowl-shaped’, or *convex*; we can simply walk downhill from our current position
3. For linear regression, we can simply *solve* for the best parameters; but in general this is not true



For complex models, it is much more difficult…

1. High-dimensional
2. Non-convex
3. No closed-form solution

…and we're never sure we've found the *best* parameters



Learning in Neural Networks

- With deep learning models, we are trying to minimize a function of many variables
- Can’t visualize it
- Can’t solve for the minimum directly
- So, we follow the slope and hope for the best (i.e. gradient descent)
- May end up in a low point that isn’t the lowest, i.e. *local minimum*
- But, if we have lots of data, things usually work out OK

Overfitting

Recall: improving one prediction may worsen another

- Suppose we predict:
  - $p_1 = .8$
  - $p_2 = .3$
  - $p_3 = .1$
- Is this a good model? Why or why not?
- Our parameters affect *all* the predictions: changing a parameter to *decrease* y_2 may also *increase* y_3

$y_1 = 1$ (dies)

$y_2 = 0$ (survives)

$y_3 = 0$ (survives)

p_1? x p_2? x p_3?

With a big a enough model, this is no longer true.

- *perfect* predictions?

  predict $p(y_i) = y_i$

- when is this possible?
  - logistic regression where $P \gg N$
  - MLP with final hidden layer of size $M \gg N$
  - models with $\gg N$ parameters

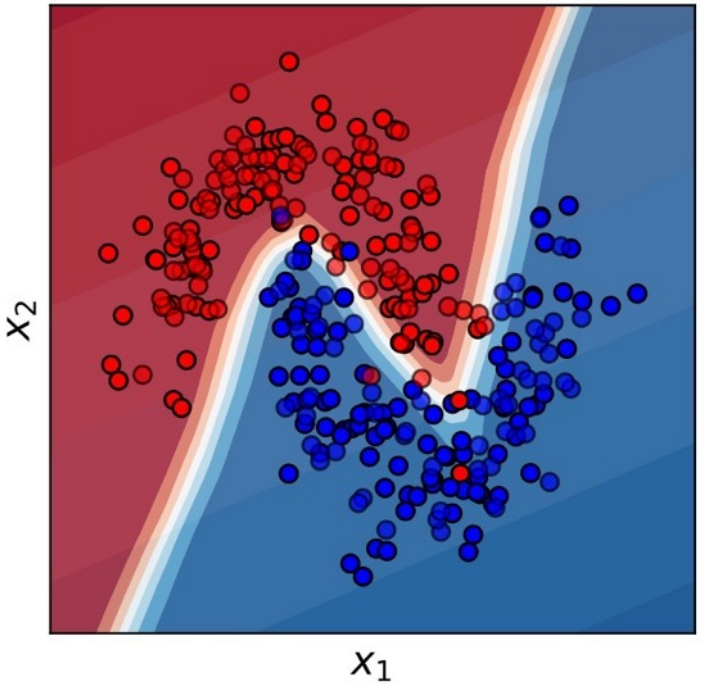
$y_1 = 1$ (dies)

$y_2 = 0$ (survives)

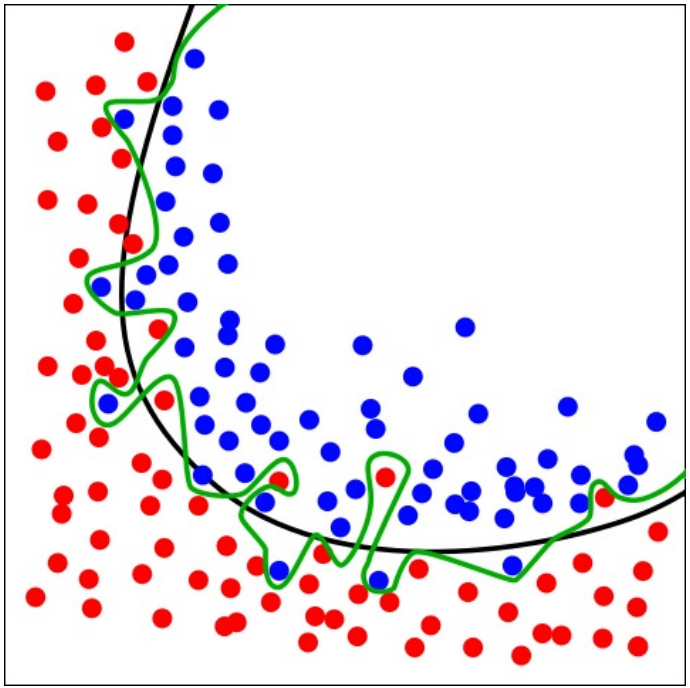
$y_3 = 0$ (survives)

$p_1?$ x $p_2?$ x $p_3?$

We like flexible, non-linear decision boundaries…



But when we start making our decision boundary arbitrarily complex to `fit' the training set… this is overfitting



Green boundary:
- Correct predictions for *all* training data
- Very likely to be overfitting

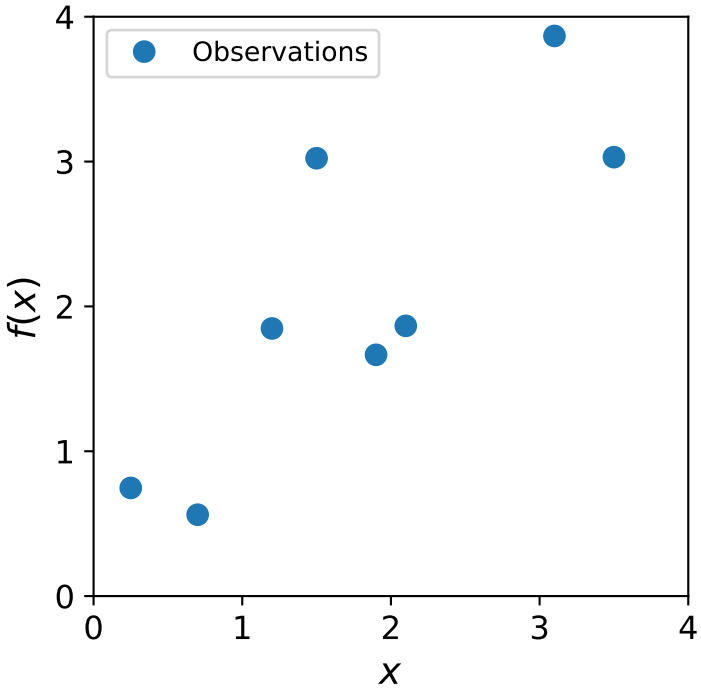
Black boundary:
- Balance between fit and model complexity

-> The black boundary is likely to perform better on new data

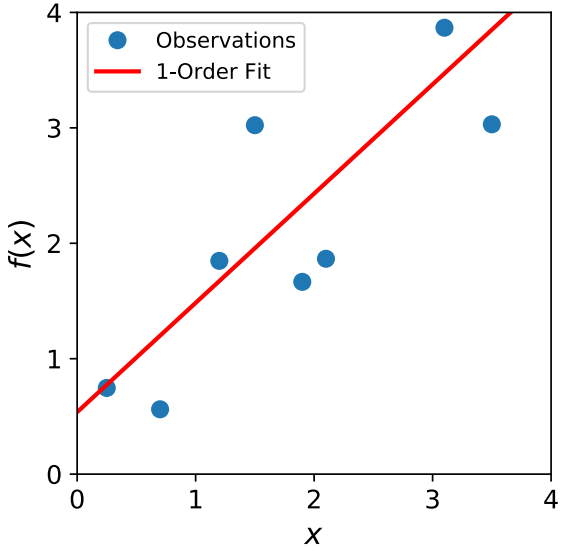
Overfitting

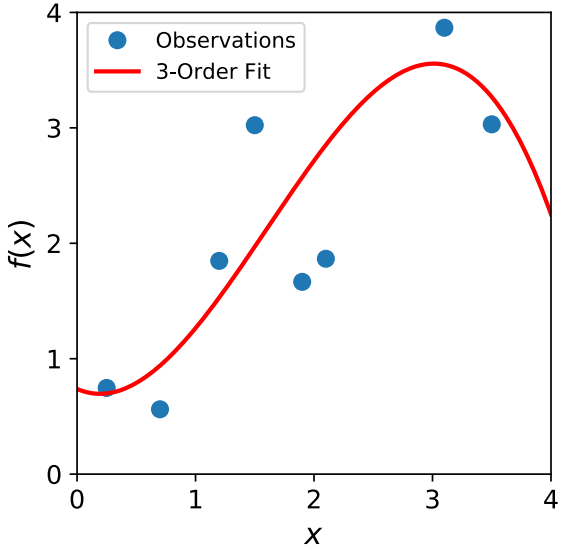
"Overfitting" happens when the learned model increases complexity to fit the observed training data *too well* – will not work to predict future data!

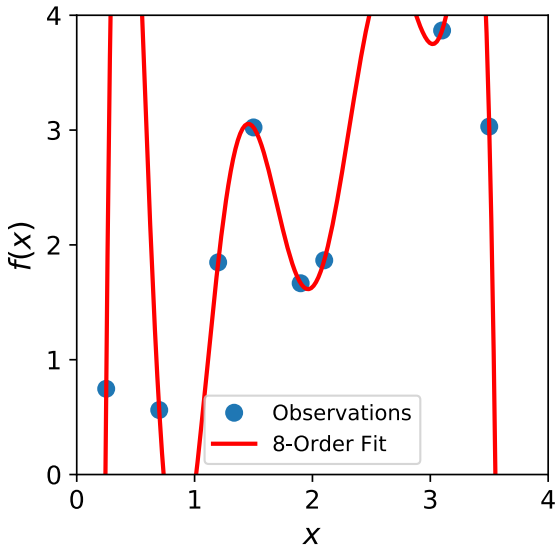
What would we want to use to fit these example data points?



Classic Example: Increasing Polynomial Order



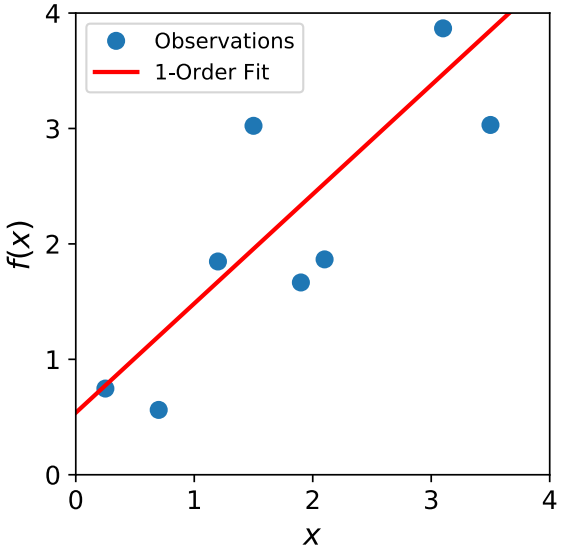




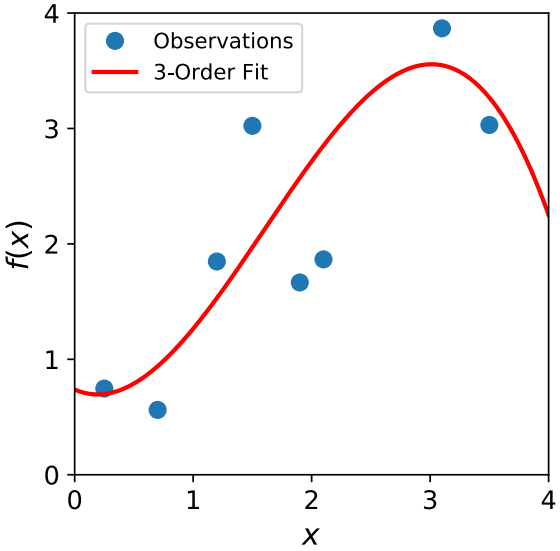
Increasing complexity does not seem appropriate…

With a complex enough model, we can typically predict our training labels perfectly.

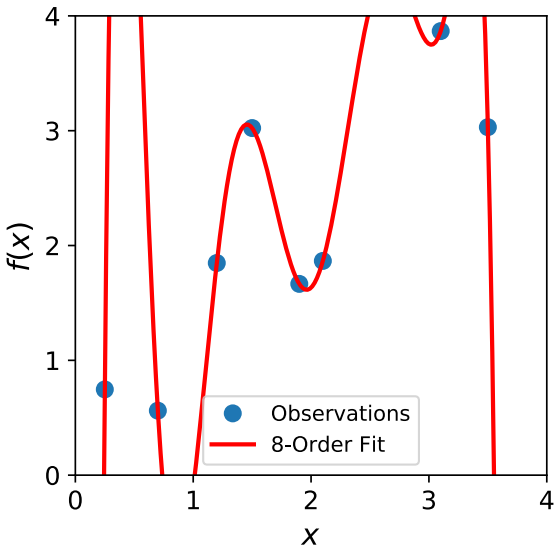
The simpler model is likely best with limited data



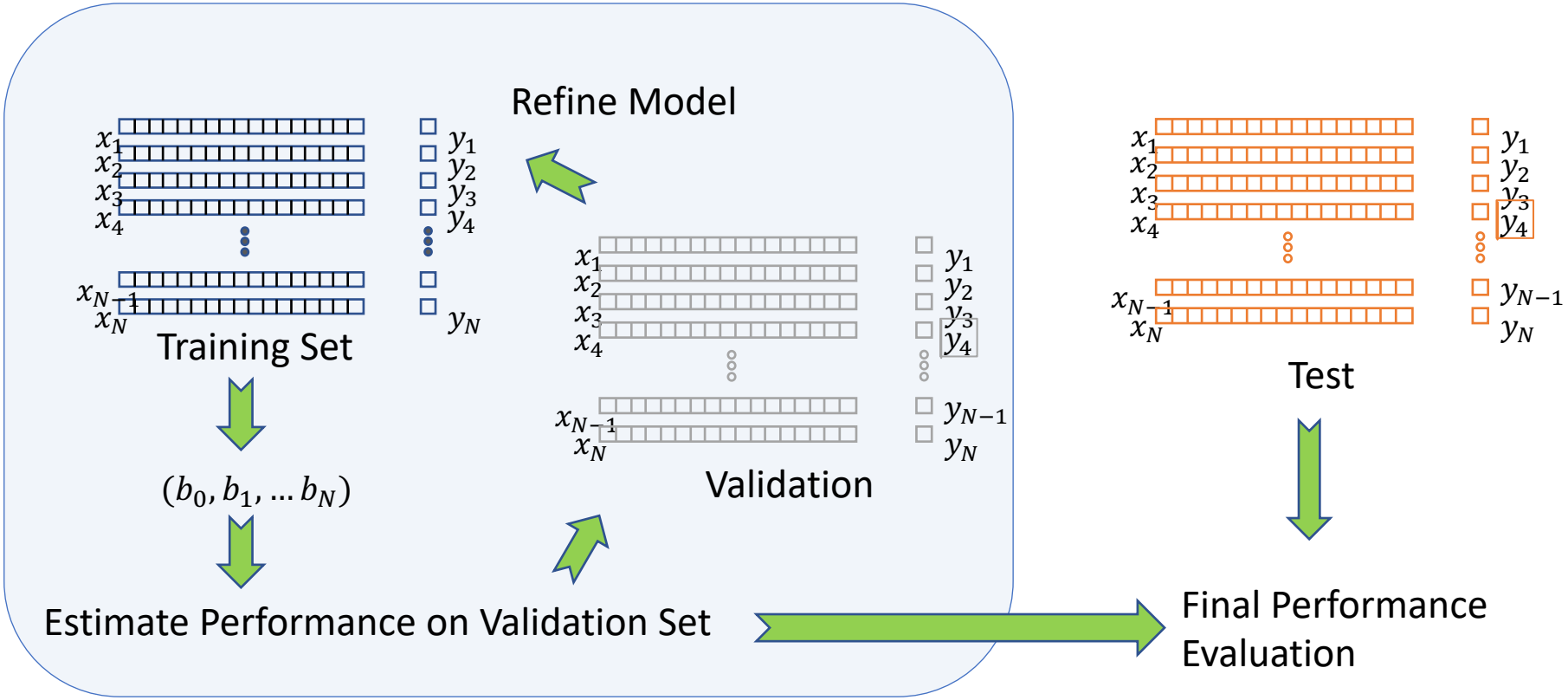
May be justified; the validation set will tell us



Perfect predictions, but unlikely to generalize

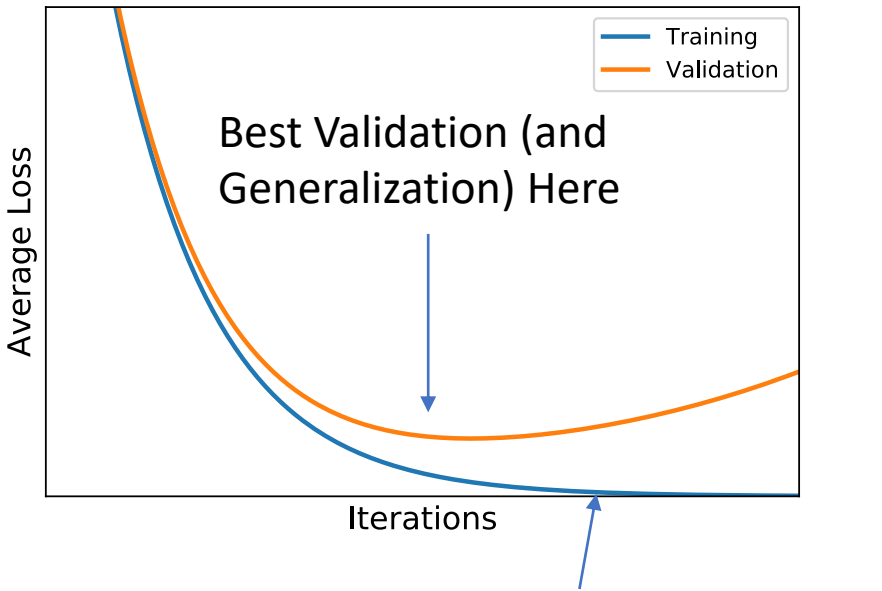


To protect against overfitting, see how well the model performs on previously unseen data.



Early Stopping

- During optimization, we can check the validation loss as we go.
- Instead of optimizing to convergence, we can optimize until the *validation* loss stops improving
  - Saves computational cost
  - Performs better on validation (and test) sets
- Widely used technique in the field



Other Ways to Use the Validation Set

- Choosing between models (e.g. MLP, LR)
- Choosing how strongly to *regularize* the model, i.e. penalize parameters
- Choosing network depth, width, etc.
- Many other “hyperparameters” we might tune

Conclusions

- *Learning* consists in setting model parameters to maximize some measure of fit (or equivalently, minimize some loss)
- This sounds easy, but is difficult in practice when working with complex models
- Greater model complexity is often, but not always, advantageous
- Proper model validation is critical to estimate real-world performance and prevent overfitting