Model Learning, Validation, and Overfitting

June 11, 2020

Lecture 3, Applied Data Science
MMCi Term 4, 2020

Matthew Engelhard

MODEL LEARNING

Learning (or *training*) our model means:

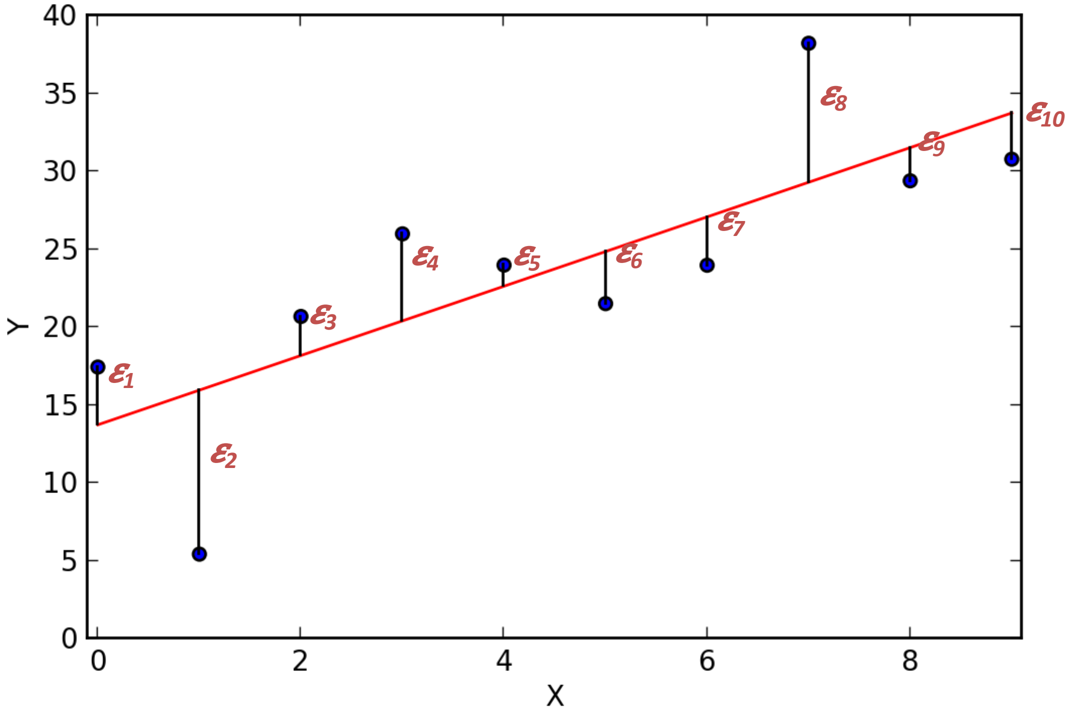
-> Setting our parameters to the specific values that are the best match for our training data

What do we mean by “best match”?

-> We set our parameters to the values that maximize some measure of fit

-> Alternatively, we set them to values that minimize a penalty (i.e. a *loss*) that we choose

Linear Regression Measure of Fit: Mean Squared Error



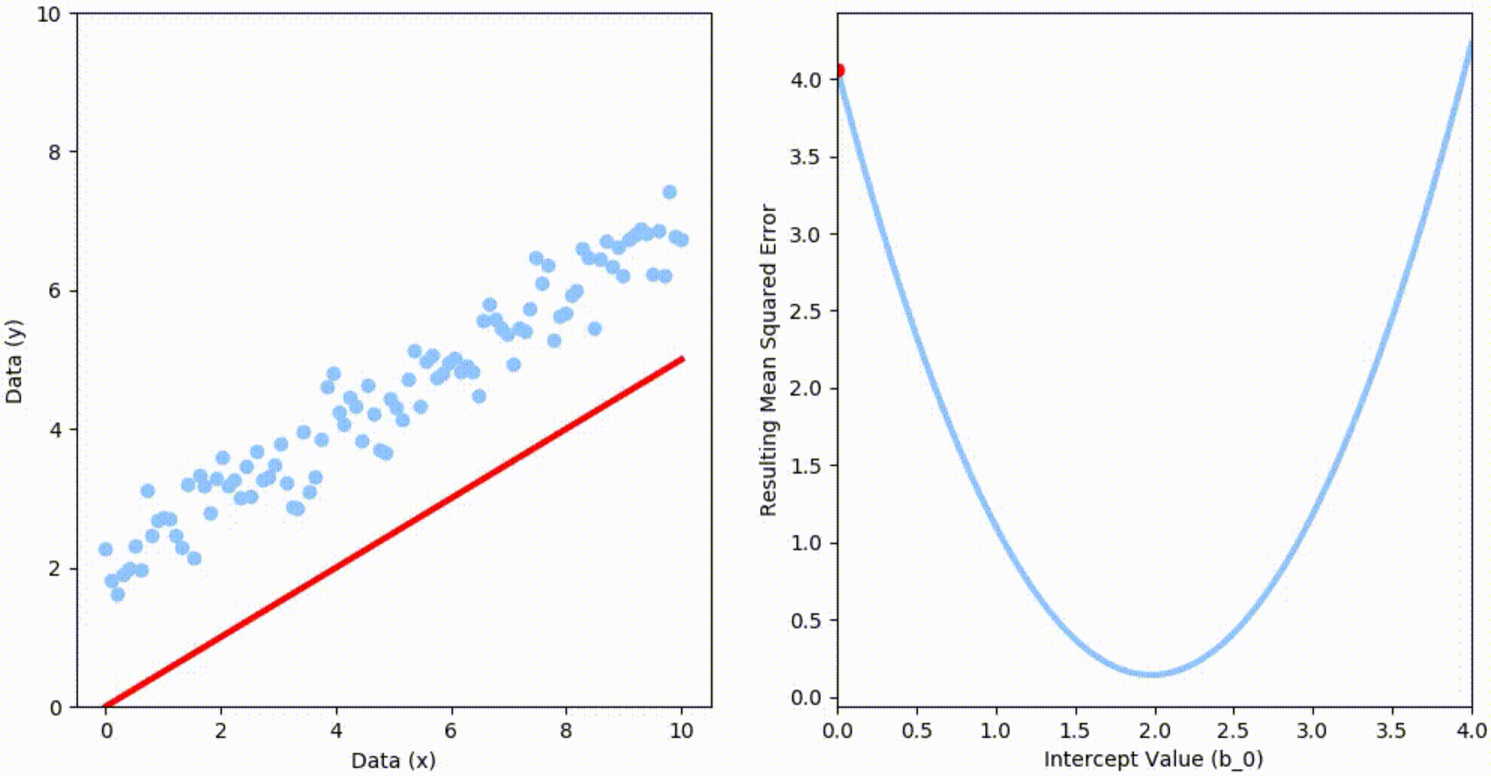
Error:

$$\varepsilon_i = y_i - \hat{y}_i$$

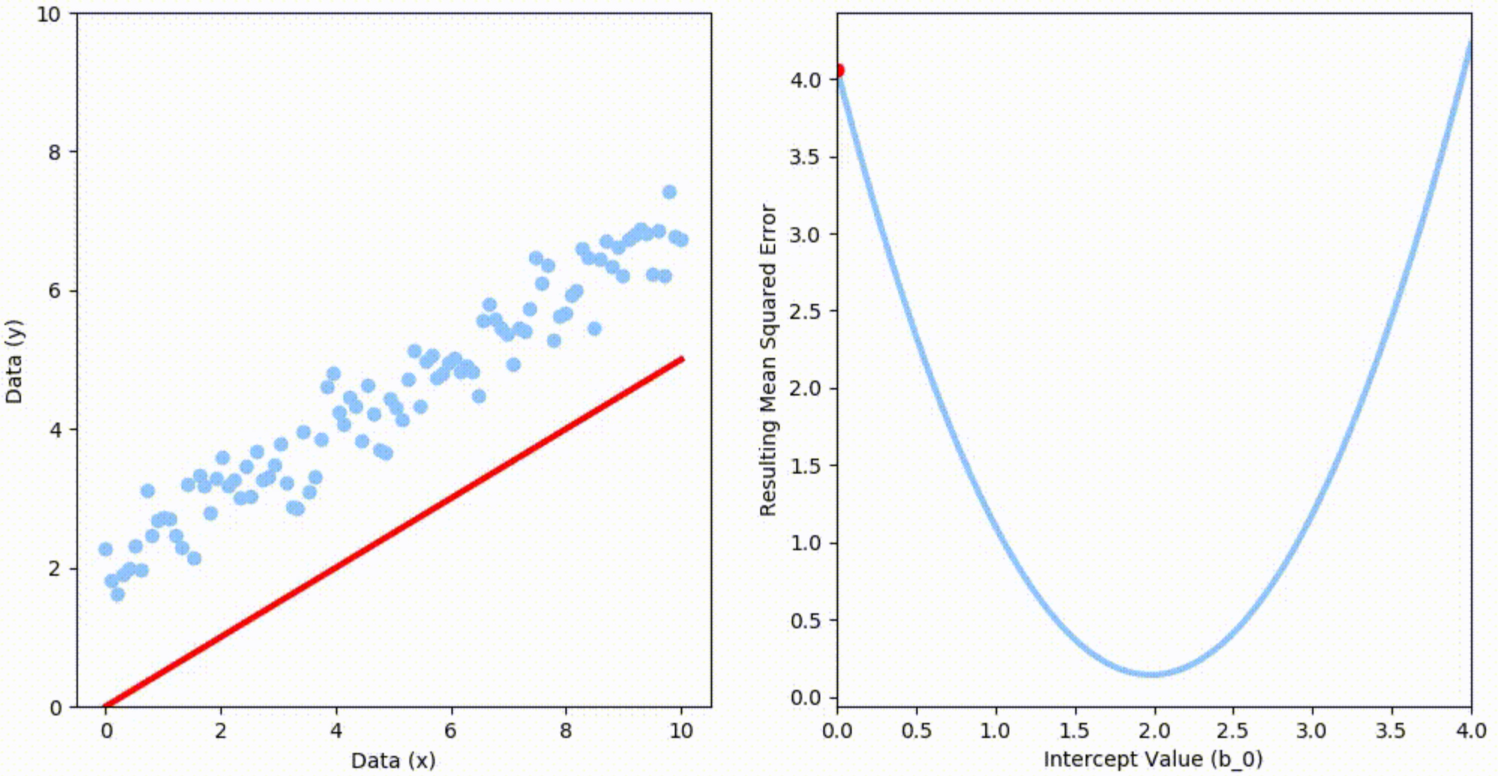
Mean square error (MSE)

$$\frac{1}{N}\sum_{i=1}^{N} \varepsilon_i^2$$

Suppose the slope is known.
What happens to the MSE as we move the intercept (b_0)?



What is the best choice of intercept (b_0) for these data, the one that minimizes the mean squared error?



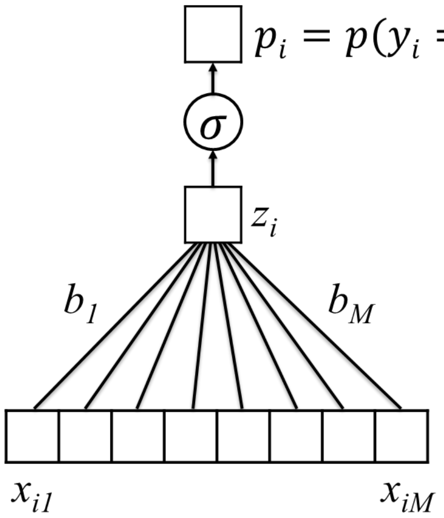
Logistic Regression Measure of Fit:

Probability that events $y_1, \dots, y_N$ would have occurred if our model were correct

Probability that event y_i would have occurred if our model were correct:

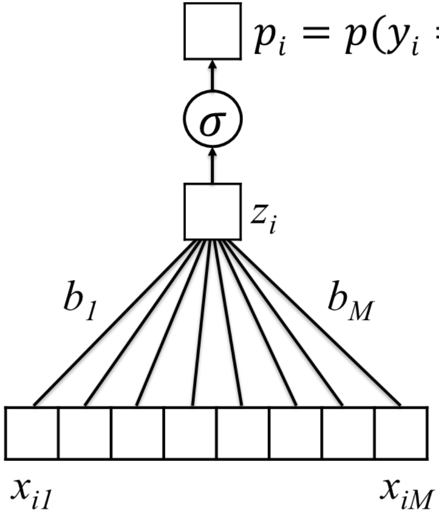
$$\left(\sigma(z_i)\right)^{y_i}\left(1-\sigma(z_i)\right)^{(1-y_i)}$$

$p_i = p(y_i = 1|x_i) = \sigma(z_i)$



Logistic Regression Measure of Fit:

Probability that events $y_1, \dots, y_N$ would have occurred if our model were correct



$$p_i = p(y_i = 1 | x_i) = \sigma(z_i)$$

Probability that event y_i would have occurred if our model were correct:

$$\underbrace{\left(\sigma(z_i)\right)^{y_i}\left(1 - \sigma(z_i)\right)^{(1-y_i)}}_{\begin{cases} p_i, & y_i = 1 \\ 1, & y_i = 0 \end{cases}}$$

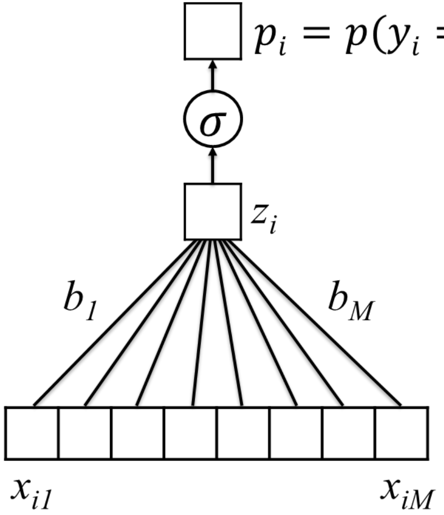
Logistic Regression Measure of Fit:

Probability that events $y_1, \ldots, y_N$ would have occurred if our model were correct

Probability that event y_i would have occurred if our model were correct:

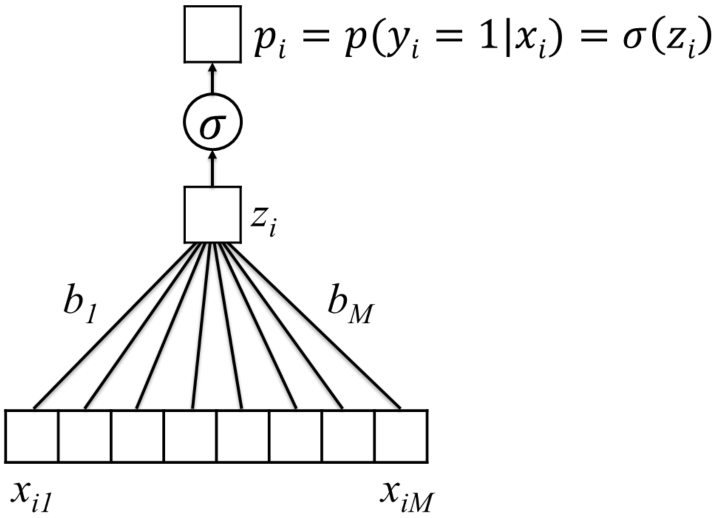
$$(\sigma(z_i))^{y_i}\underbrace{(1-\sigma(z_i))^{(1-y_i)}}_{\begin{cases} 1, & y_i = 1 \\ 1-p_i, & y_i = 0 \end{cases}}$$

$p_i = p(y_i = 1 | x_i) = \sigma(z_i)$



Logistic Regression Measure of Fit:

Probability that events $y_1, \ldots, y_N$ would have occurred if our model were correct



Probability that event y_i would have occurred if our model were correct:

$$\left(\sigma(z_i)\right)^{y_i}\left(1-\sigma(z_i)\right)^{(1-y_i)}$$

Probability that events $y_1, \ldots, y_N$ would have occurred if our model were correct:

$$\prod_{i=1}^{N}\left(\sigma(z_i)\right)^{y_i}\left(1-\sigma(z_i)\right)^{(1-y_i)}$$

Logistic Regression Measure of Fit:

Probability that events $y_1, \dots, y_N$ would have occurred if our model were correct

Maximize the probability that events $y_1, \dots, y_N$ would have occurred if our model were correct:

$$\prod_{i=1}^{N} \left(\sigma(z_i)\right)^{y_i} \left(1 - \sigma(z_i)\right)^{(1-y_i)}$$

Easier to maximize the log of this quantity (i.e. the log-likelihood):

$$\sum_{i=1}^{N} y_i \log \sigma(z_i) + (1 - y_i) \log\left(1 - \sigma(z_i)\right)$$

Maximize the log-likelihood = minimize the "cross-entropy" loss

Log-likelihood:

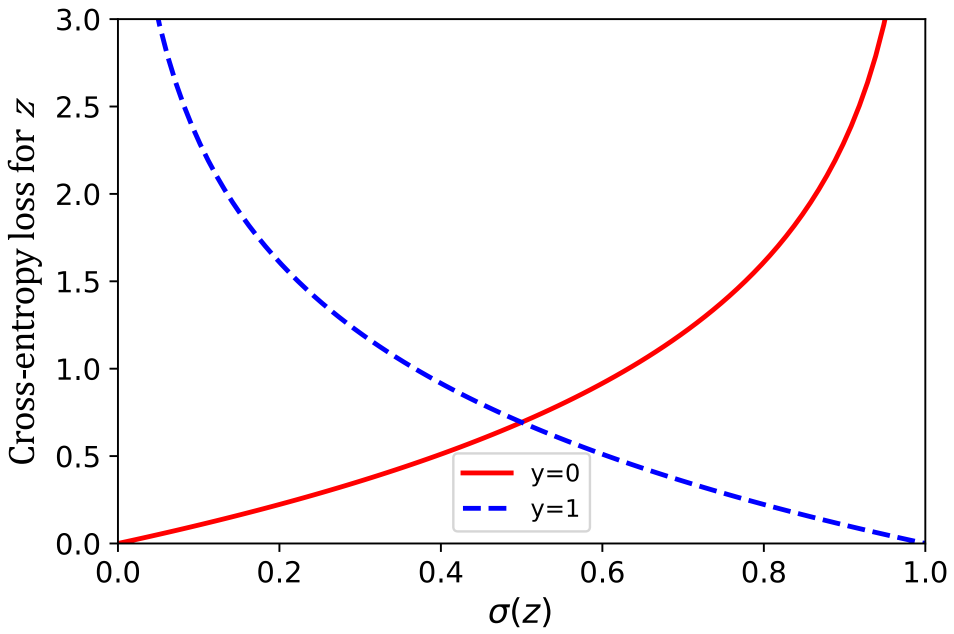
$$\sum_{i=1}^{N} y_i \log \sigma(z_i) + (1 - y_i) \log\big(1 - \sigma(z_i)\big)$$

Maximize the log-likelihood = minimize the "cross-entropy" loss

Cross-entropy loss:

$$\sum_{i=1}^{N} -y_i \log \sigma(z_i) - (1-y_i)\log\big(1-\sigma(z_i)\big)$$

The cross-entropy loss is also used for other classification models, including convolutional neural network classifiers for image processing.



How do we minimize the loss?

- The cross-entropy loss just tells us *what* quantity we should be minimizing
- In some cases (e.g. linear regression), we can solve for the minimum directly
- But, we'd like to have an approach that works even for very complex models

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	[redacted]				

PERFORMANCE

accuracy	avg_loss
0.65	0.3010

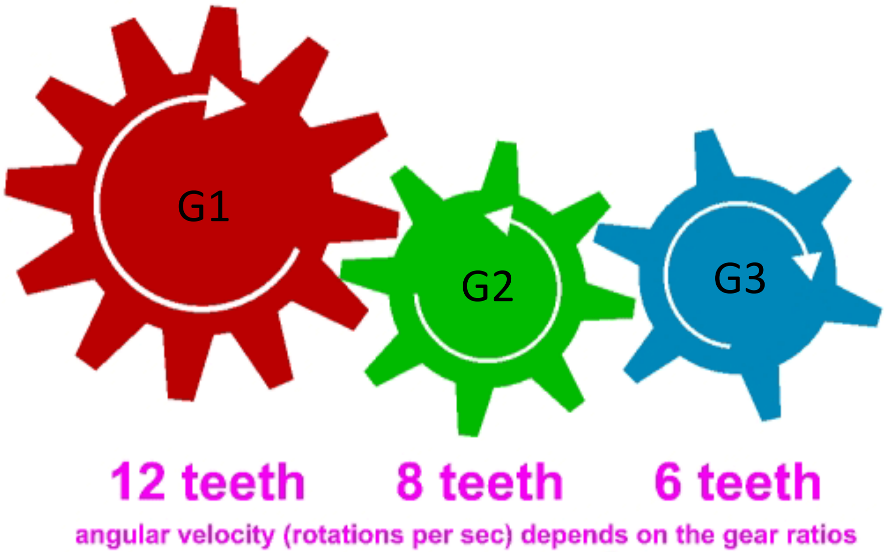
Strategy: determine how small changes in parameters affect the loss

<u>We know:</u>

- 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

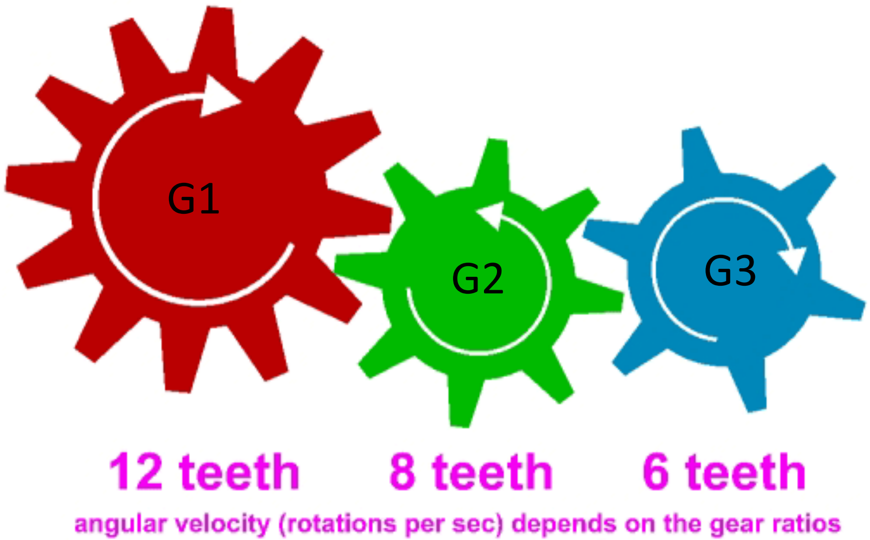
<u>We know:</u>

- 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.

➢ $\left(-\frac{12}{8}\right) * \left(-\frac{8}{6}\right) = \frac{12}{6} = 2$



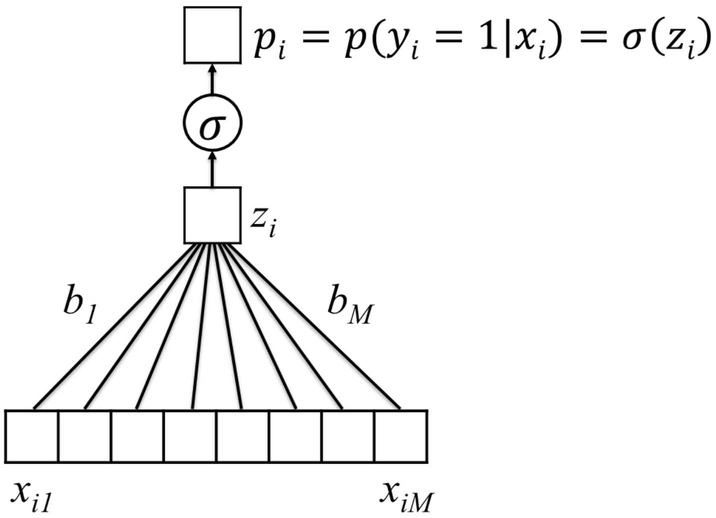
Strategy: determine how small changes in parameters affect the loss

<u>We know:</u>

- 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)?

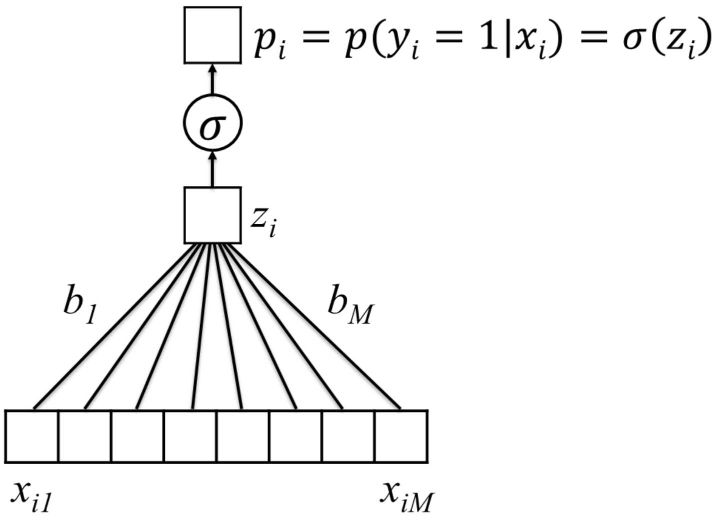
➢ Multiply the effects.



Strategy: determine how small changes in parameters affect the loss

<u>This is called the *chain rule* (calc 101)</u>

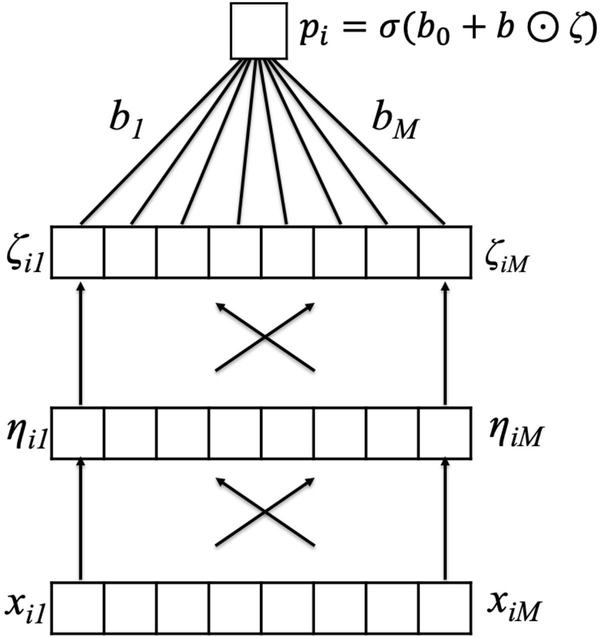
- We use it to see how small changes in parameters affect the loss
- Could be a very long chain…
- Some parameters have a greater effect than others
- We change all parameters at once, with each change proportional to that parameter's effect on the loss
    - This is *gradient descent*



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



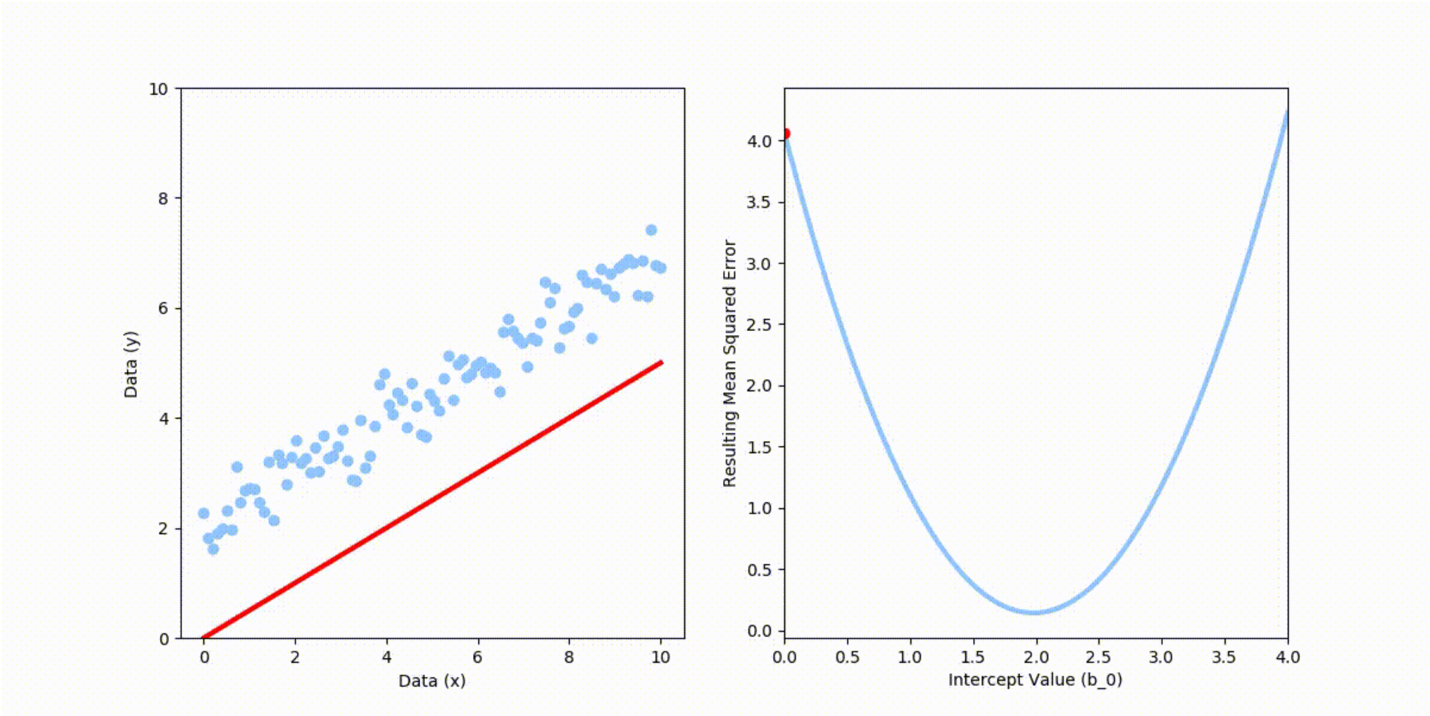
Learning: find parameters that minimize the loss

Let's begin by finding the value of a *single* parameter that minimizes the loss. We'll consider the intercept b_0 of a linear regression model.

$$y_i = bx_i + b_0 + \varepsilon_i$$

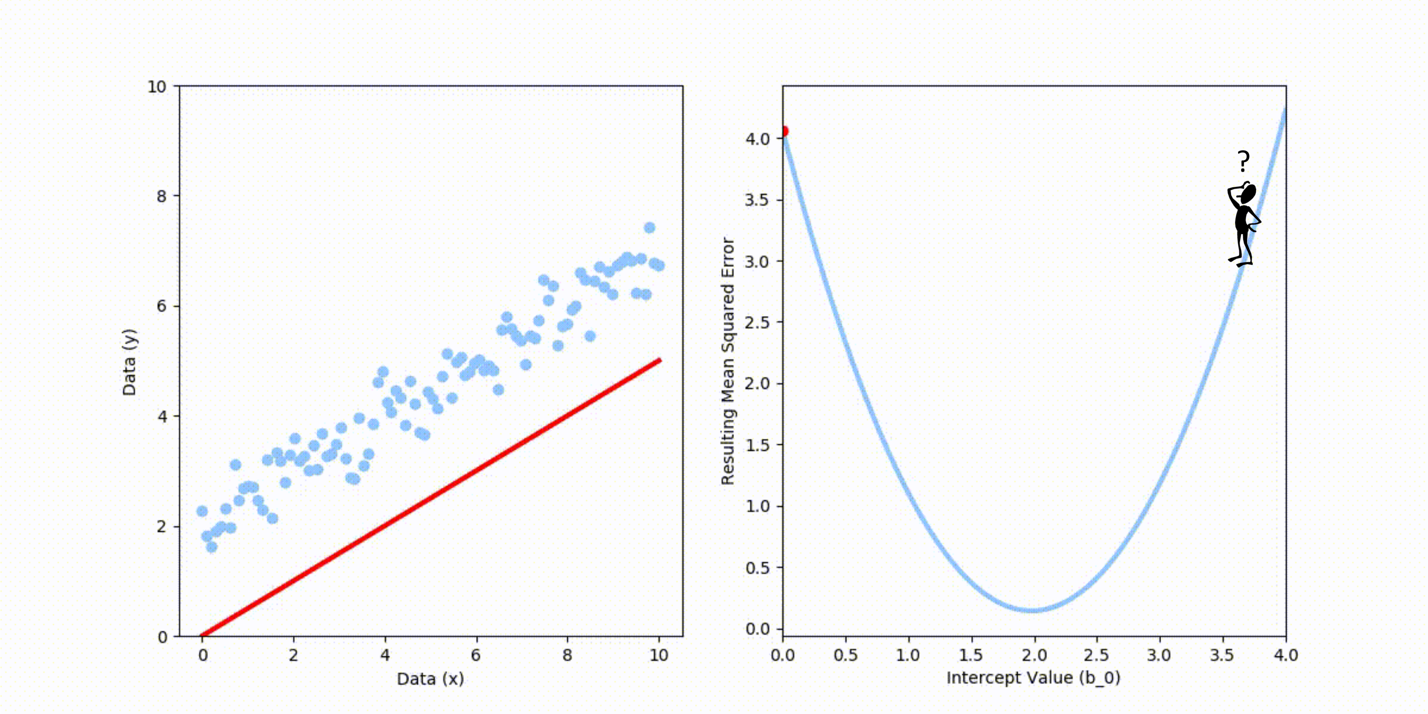
Learning: find parameters that minimize the loss

Easy!!



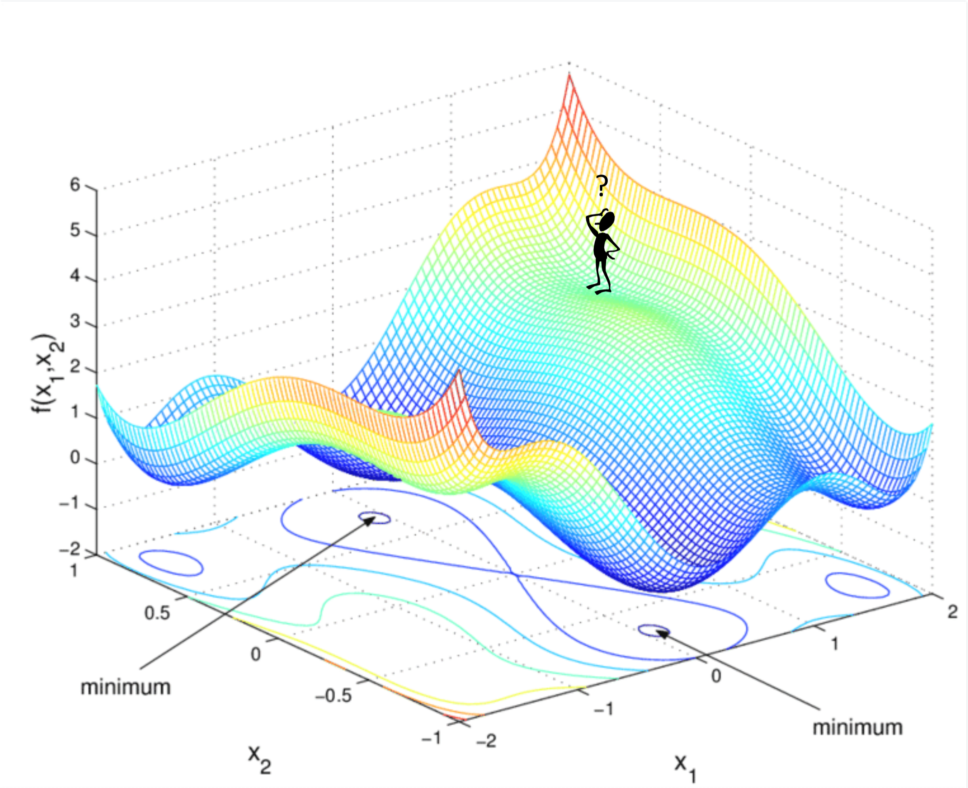
Learning: find parameters that minimize the loss

Easy!!



Learning: find parameters that minimize the loss

Easy?



Learning: find parameters that minimize the loss

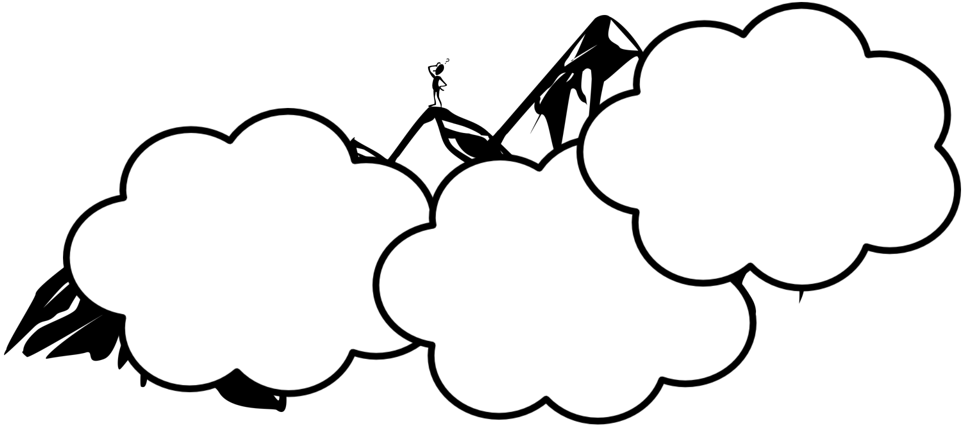
Not easy.

Learning: find parameters that minimize the loss

Not easy.

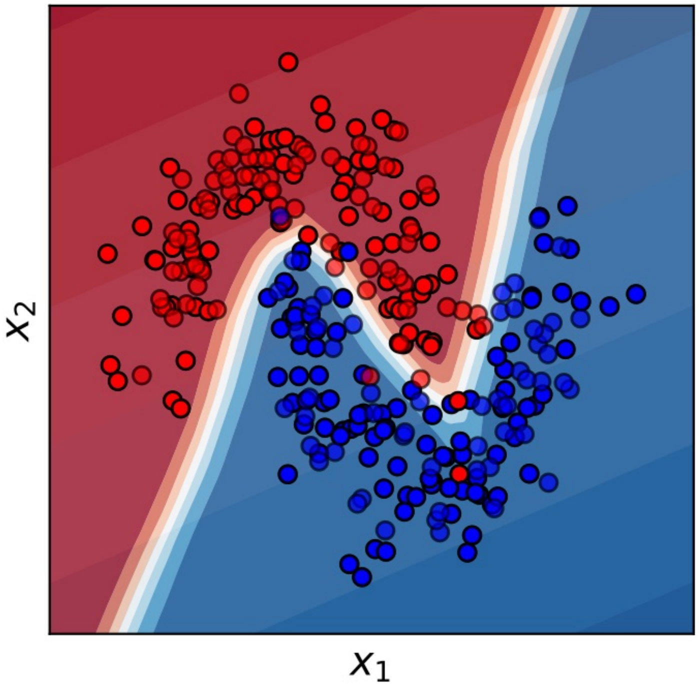
Learning: find parameters that minimize the loss

- 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

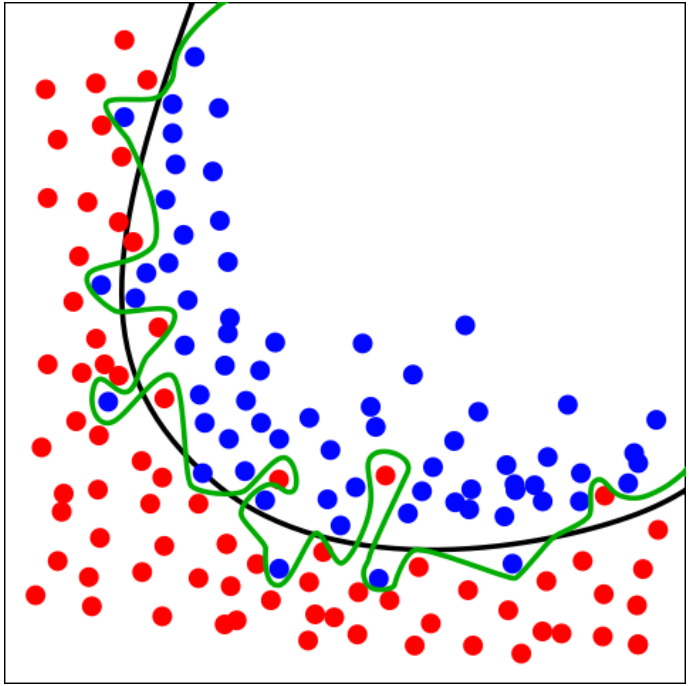


OVERFITTING

We like flexible, non-linear decision boundaries…



But some models can be *too* flexible.



By Chabacano - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=3610704

Green boundary:
- This is overfitting

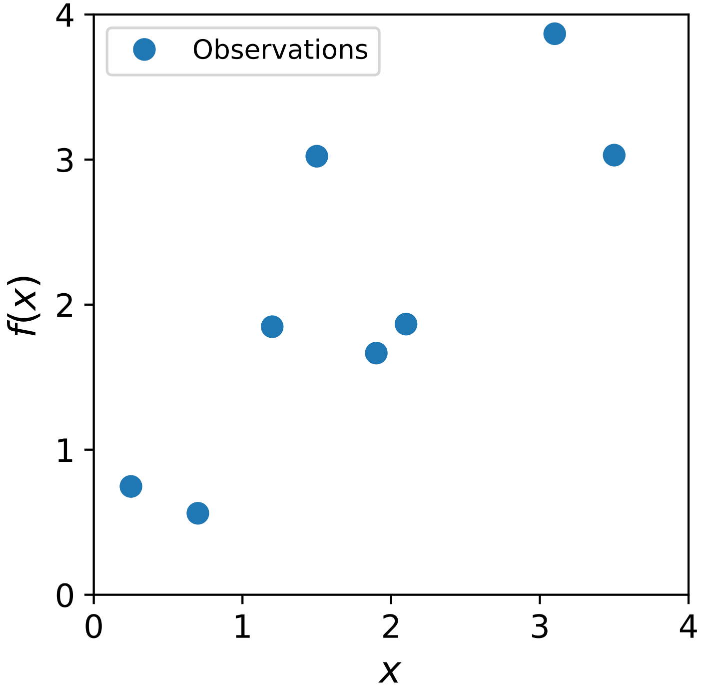
Black boundary:
- Balance between fit and model complexity

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

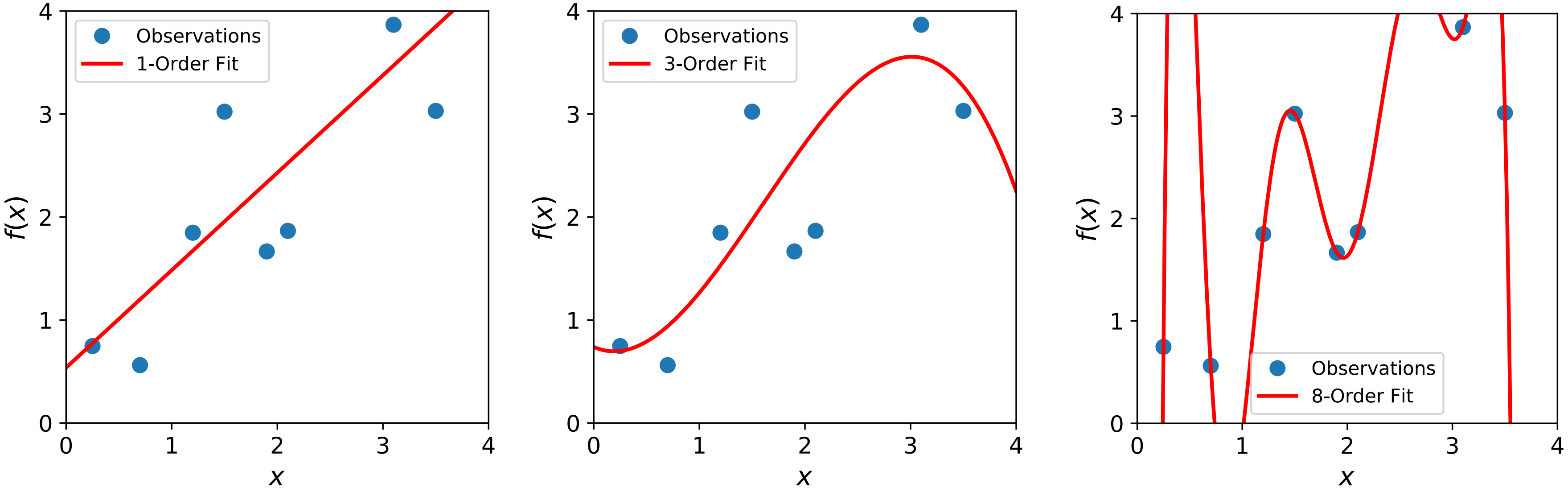
Overfitting

"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?

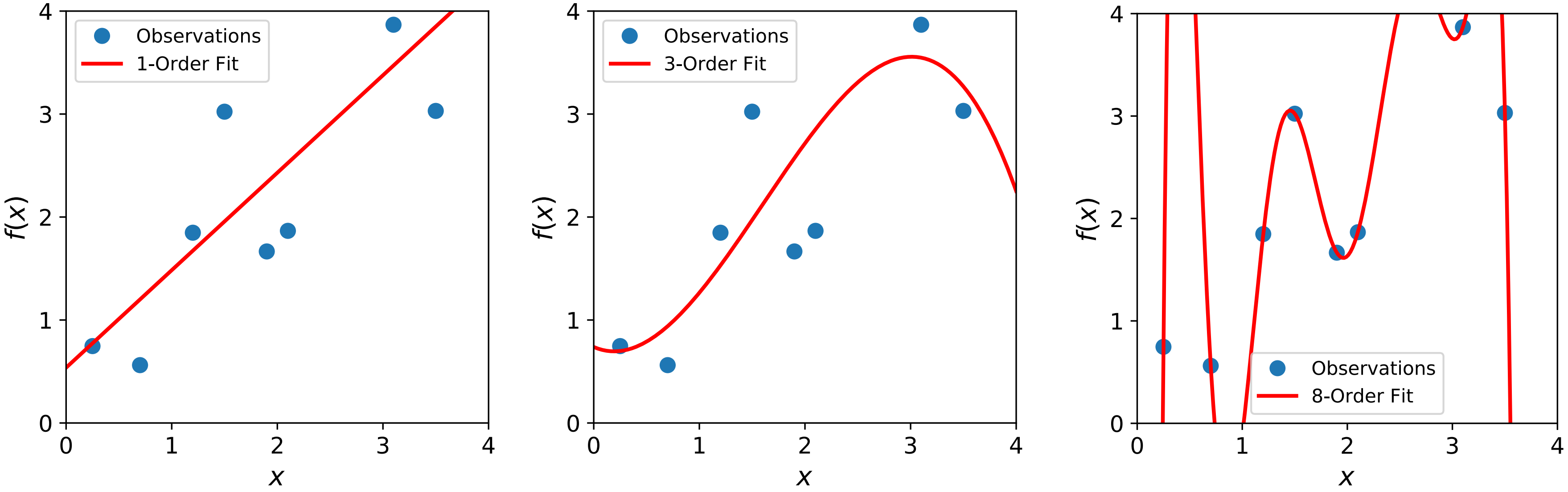


Classic Example: Increasing Polynomial Order



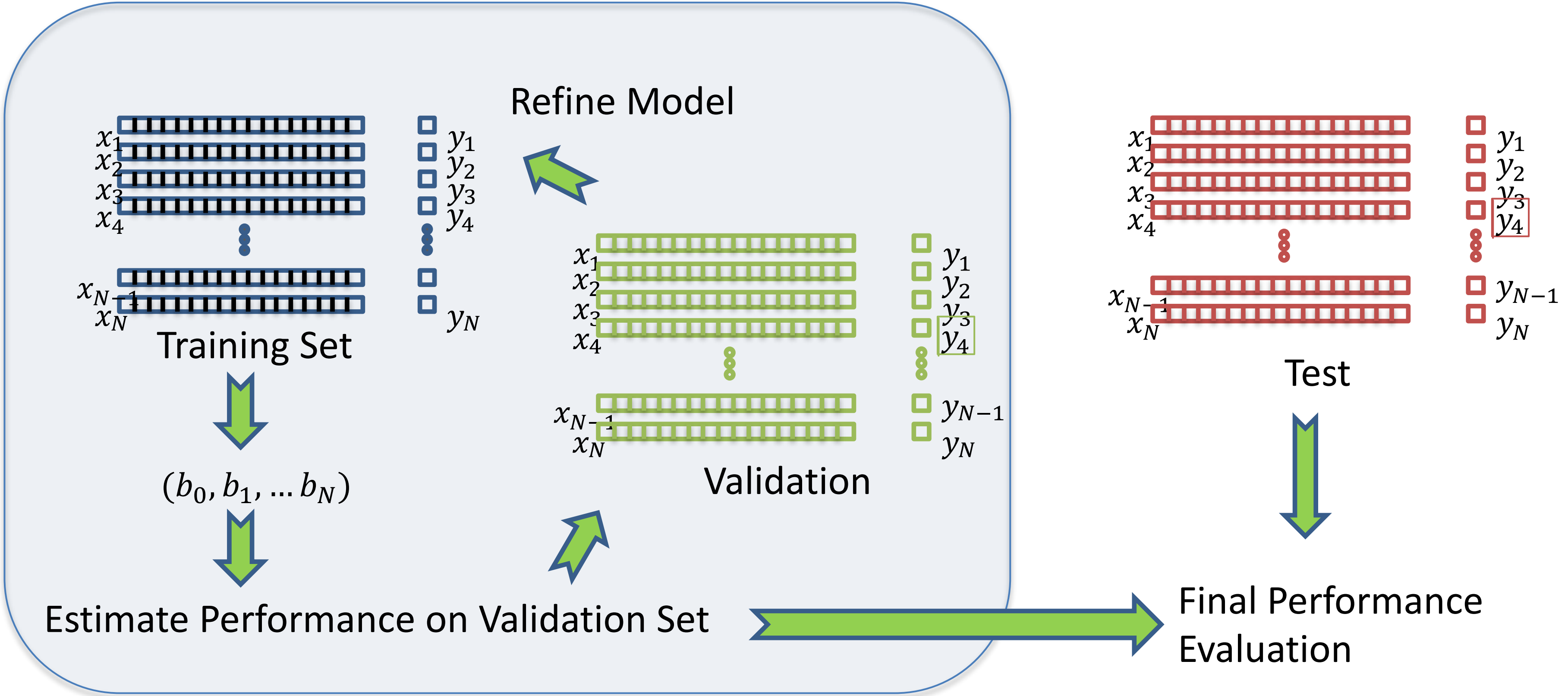
Increasing complexity does not seem appropriate…

With a flexible enough model, we can typically reach 100% accuracy on our training set



Increasing complexity does not seem appropriate…

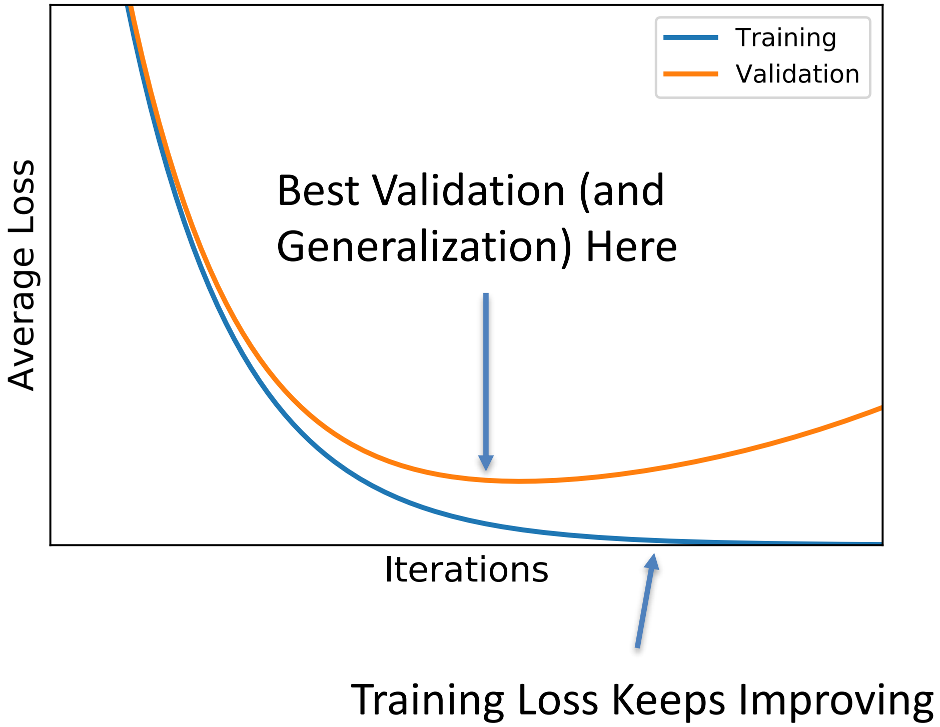
Using a Test Set Protects Against Overfitting: See how the model performs on *new* data



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