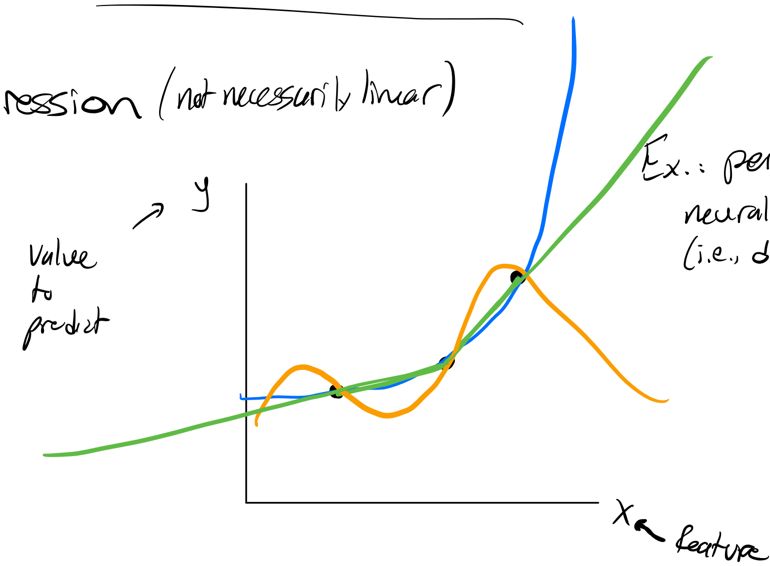
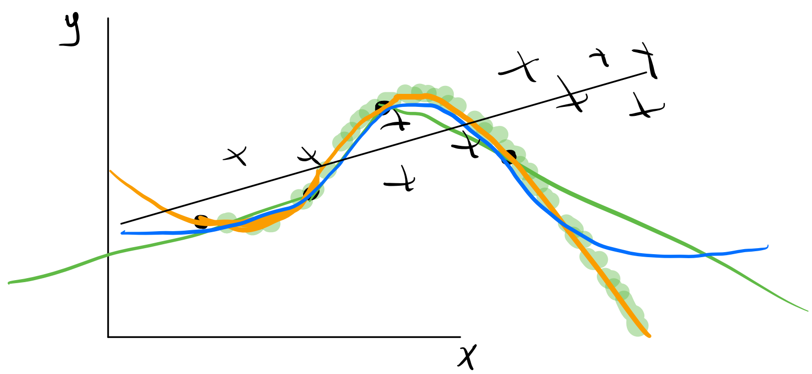
Occam's Razor

Task: regression (not necessarily linear)

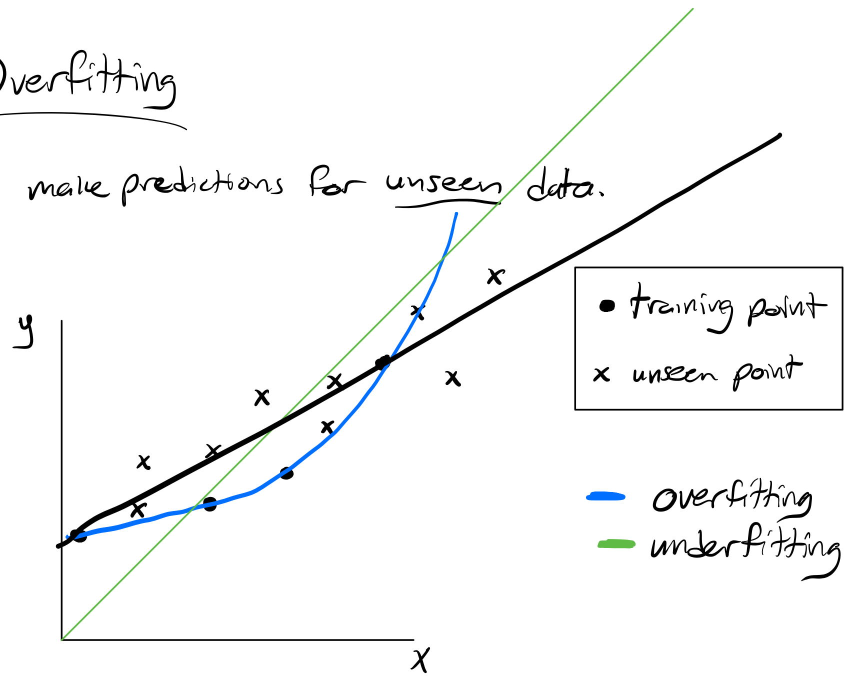


Ex.: perform biological neural network regression (i.e., do it yourself!)



Overfitting

Goal: make predictions for unseen data.



overfitting:

model mistakes noise for signal

underfitting

model doesn't (can't) fit signal

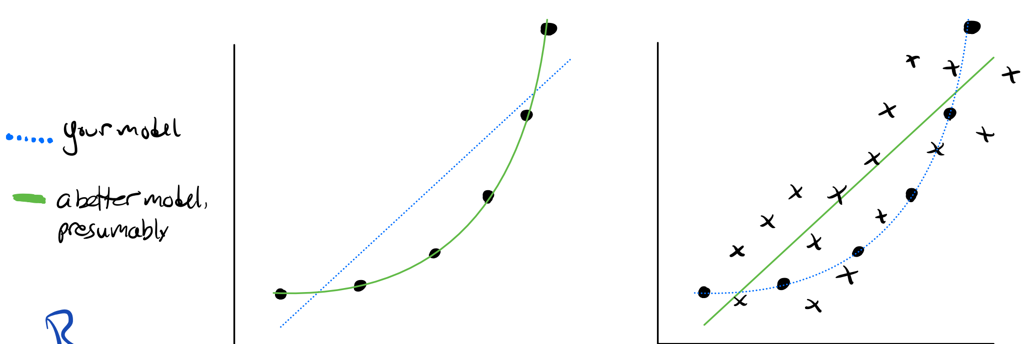
"Simplest possible explanation for the data" needs to take into account noise/sampling error

$$y = ax^2 + bx + c$$
$$a = 0$$

Bias vs. Variance

Bias: modeling error - your model can't fit the underlying phenomenon

Variance: sampling error - your model mistakes noise for the underlying phenomenon



B

V

Ex.: Which is which?

Tools in the fight against overfitting

How can we fit a model and convince ourselves it isn't overfitting?

1. Withhold data!
2. Use a simple model regardless

1. Data Splits

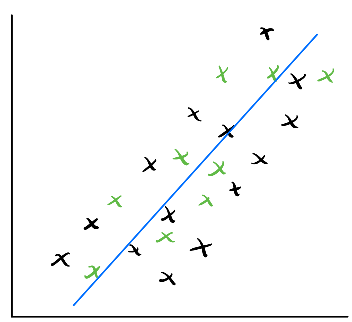
Idea: Hold back some training data

Train on ⓧ, validate on ⓧ

x training set
x validation set

Scenario: accuracy is measured as distance from x to /

All available training data:



		Accuracy on validation set	
		Bad	Good
Accuracy on training set	Bad	underfit	lucky/cheating
	Good	overfit	:)

Pseudocode for MLBot 1.0:

```
make modeling assumptions
While True:
    train
    validate        ← no longer unseen!
    if train == val == good:
        break
else:
    revisit modeling assumptions
```

Problem?

Solution? Hold out another set of available data:

test set

Use only once to see how model does on truly unseen data.

Data split best practice:

Labeled data

Train	Val	Test

What %? Depends on size of data and amount of noise.

Smaller → higher variance

larger → less data for other splits

For small data, take full advantage of as much data as possible:

Val_1	Val_2	Val_3	...	Val_k	Test

"k-fold cross-validation":

train on each subset of $k-1$ chunks, val on the last

avg val accuracy across all k trials

\+ better training, lower-variance val accuracy

\- need to train k times

"leave-one-out cross-validation":

$k = n$

Tools in the fight against overfitting

2. Regularization: build a "simplicity prior" into your model.