

# Mitigating Overfitting

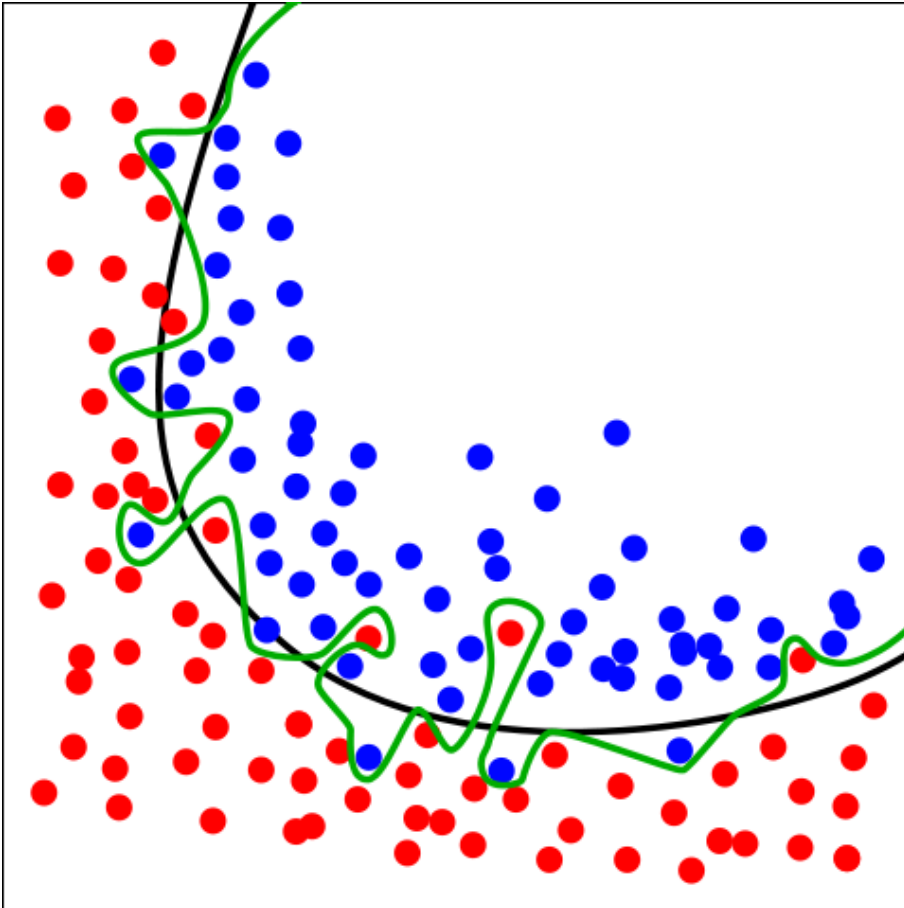
Matthew Engelhard

# Recall the NDP

- Complex models are capable of dramatically overfitting the data
- We can think of this as “memorizing” the data
- It is therefore **critical** to evaluate on a held-out test set  
(supposing what you care about is out-of-sample performance)



# Which boundary performs better on unseen data?



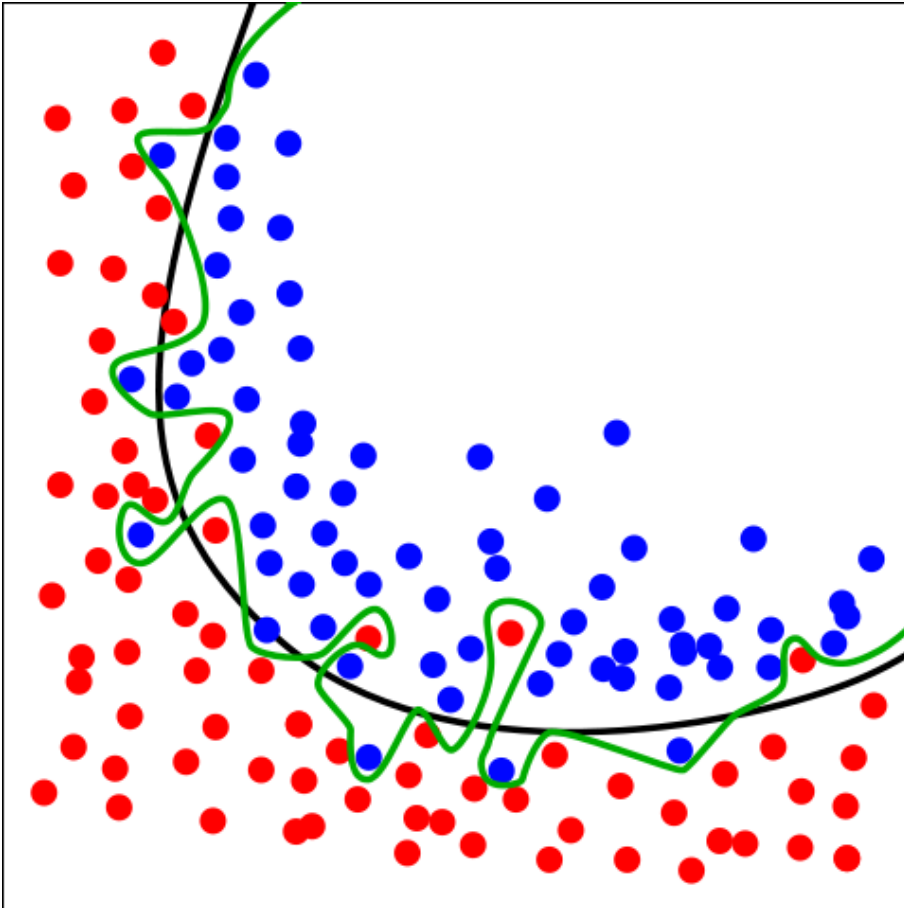
Green boundary:

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

Black boundary:

- Balance between fit and model complexity

# Quantifying Overfitting



Which boundary performs better when applied to new data? And by how much?

- We can quantify overfitting as the difference in performance (usually the cross-entropy loss) between the training and validation sets
- Greater overfitting usually (but not always) means worse out-of-sample performance
- So, we almost always take steps to mitigate it

# Let's teach our robot to identify beaches.



We show it a bunch of images and ask:

*What do you see?*

It starts listing features from most to least obvious.





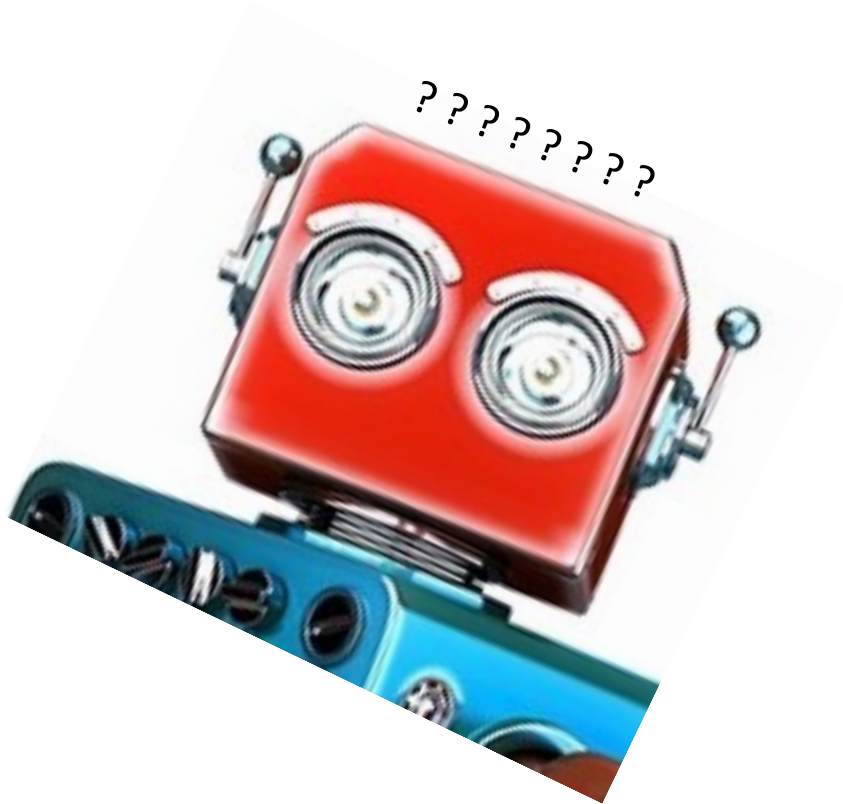
# Let's teach our robot to identify beaches.



*I think beaches have:*

- water
- land
- sky
  
- sand
- palm trees
- clouds
- ripples
- foam
  
- white sand
- clear water
- deep blue sky
- exactly 0 or 1 seashells
- Instagram color palette

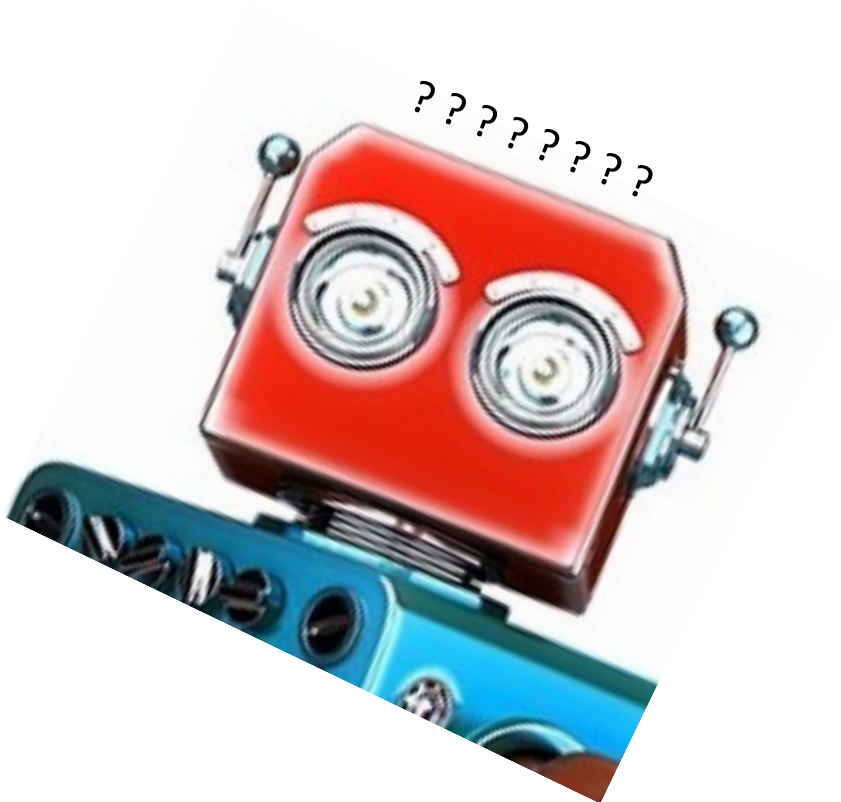
# It just doesn't know when to stop.



*I think beaches have:*

- water
- land
- sky
  
- sand
- palm trees
- clouds
- ripples
- foam
  
- white sand
- clear water
- deep blue sky
- exactly 0 or 1 seashells
- Instagram color palette

# We need to give it a strategy to decide how far to go.



*I think beaches have:*

Sure.

- water
- land
- sky

Maybe?

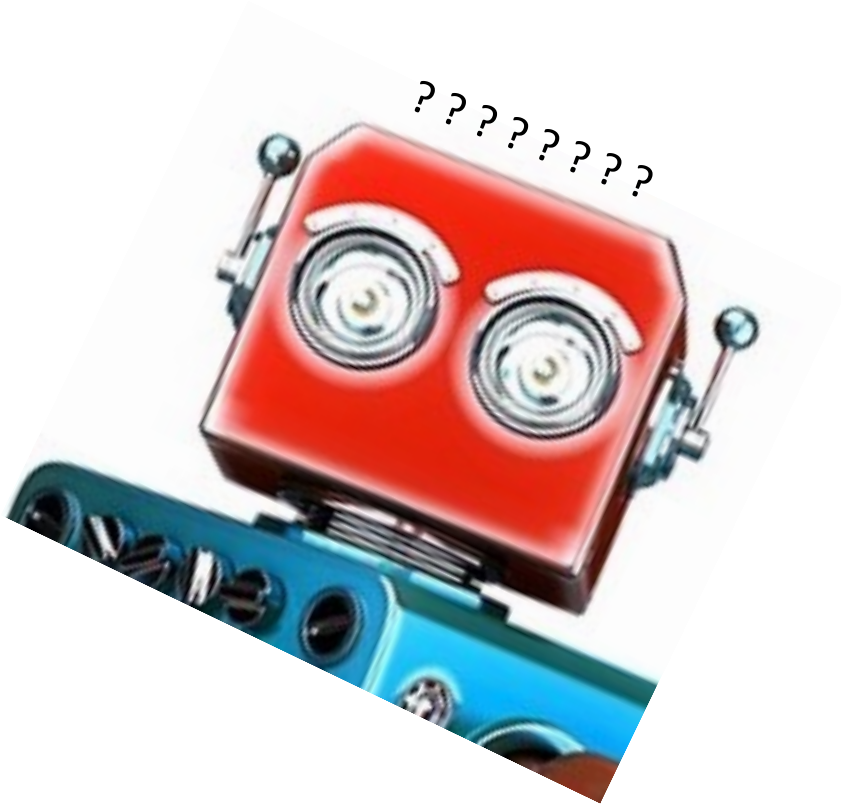
- sand
- palm trees
- clouds
- ripples
- foam

Uh... no.

- white sand
- clear water
- deep blue sky
- exactly 0 or 1 seashells
- Instagram color palette



# Strategy 1: Use as few features as you can.



*I think beaches have:*

Sure.

- water
- land
- sky

Maybe?

- sand
- palm trees
- clouds
- ripples
- foam

Uh... no.

- white sand
- clear water
- deep blue sky
- exactly 0 or 1 seashells
- Instagram color palette

Penalize: loss, but also total parameter magnitude

# Strategy 2: Keep checking other data



Training set



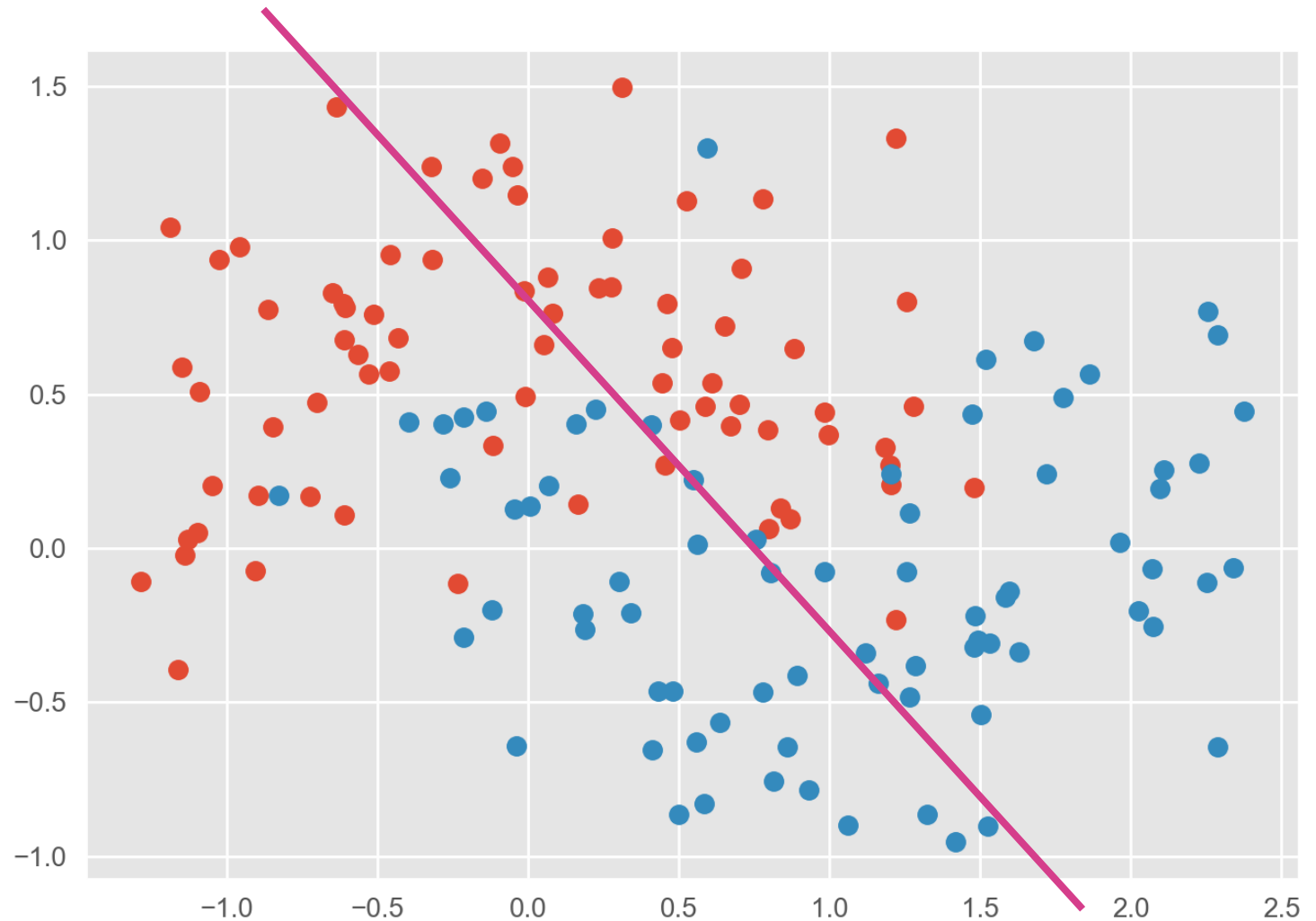
Validation set



# Let's teach a neural network to identify cancer

Initial decision boundary

(epoch 0)

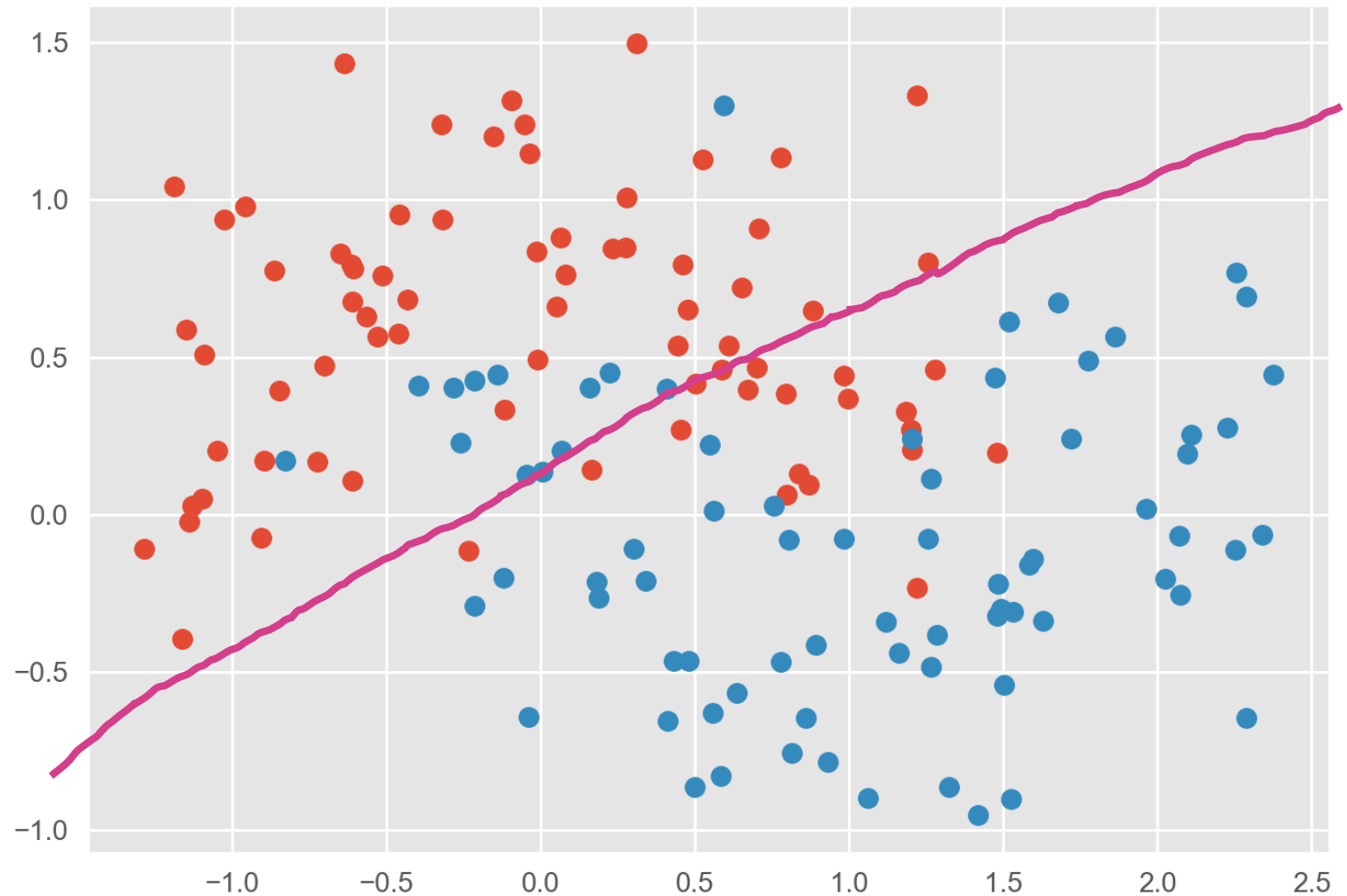


# Let's teach a neural network to identify cancer

Updated decision boundary

(epoch 1)

Sure.

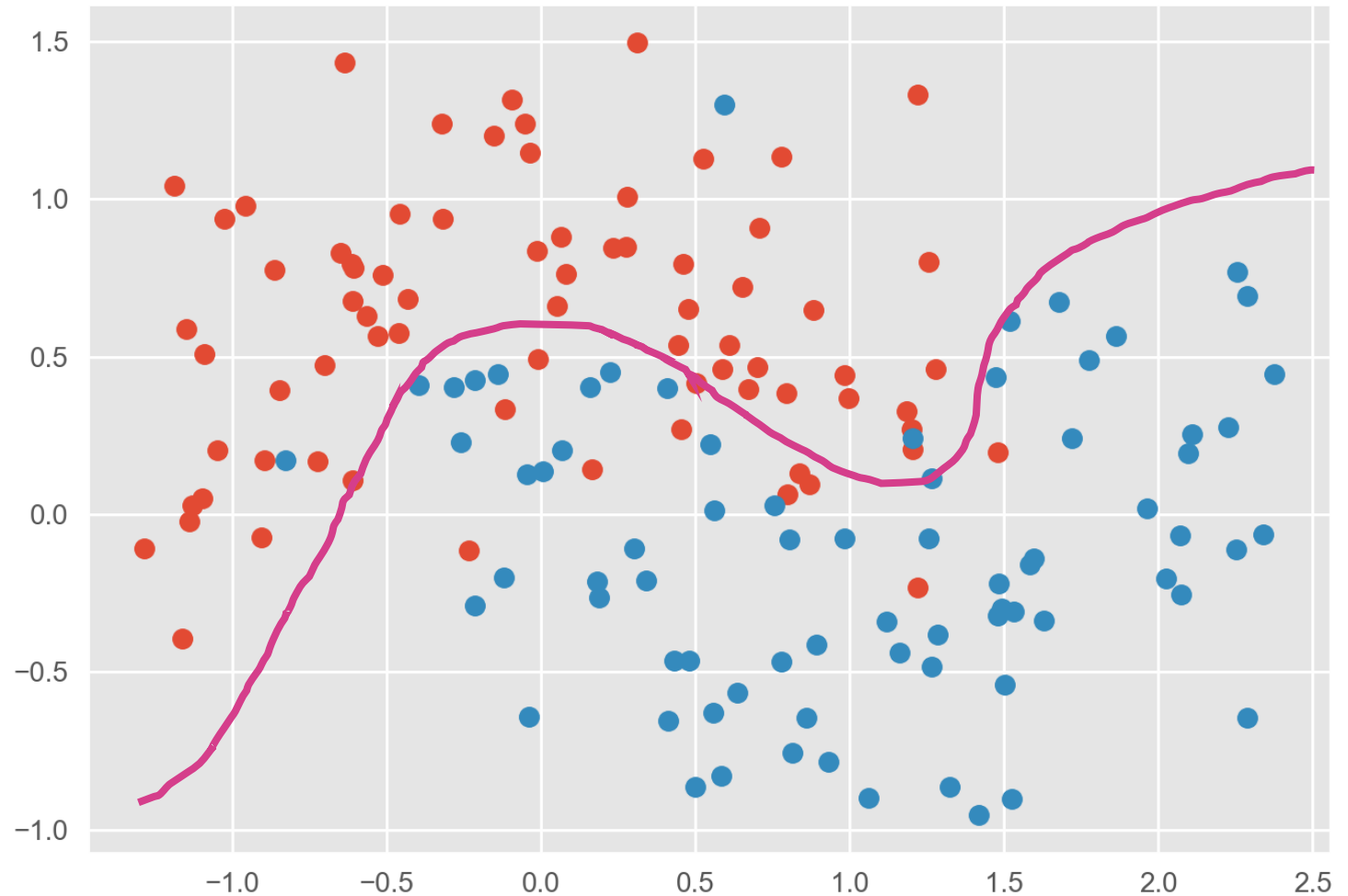


# Let's teach a neural network to identify cancer

Updated decision boundary

(epoch 4)

Wow!





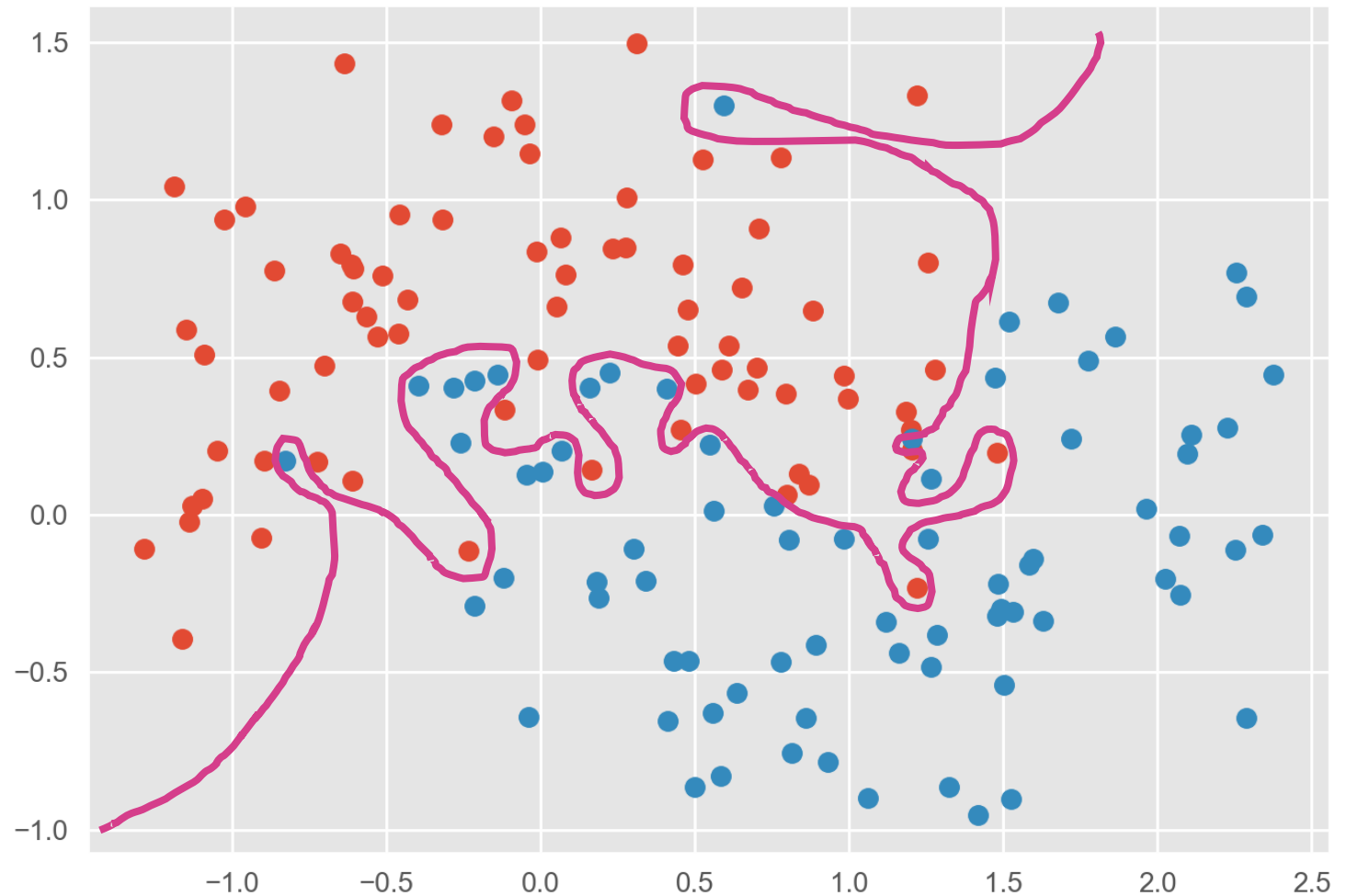
# Let's teach a neural network to identify cancer

Updated decision boundary

(epoch 23)

Uh... let's dial it back.

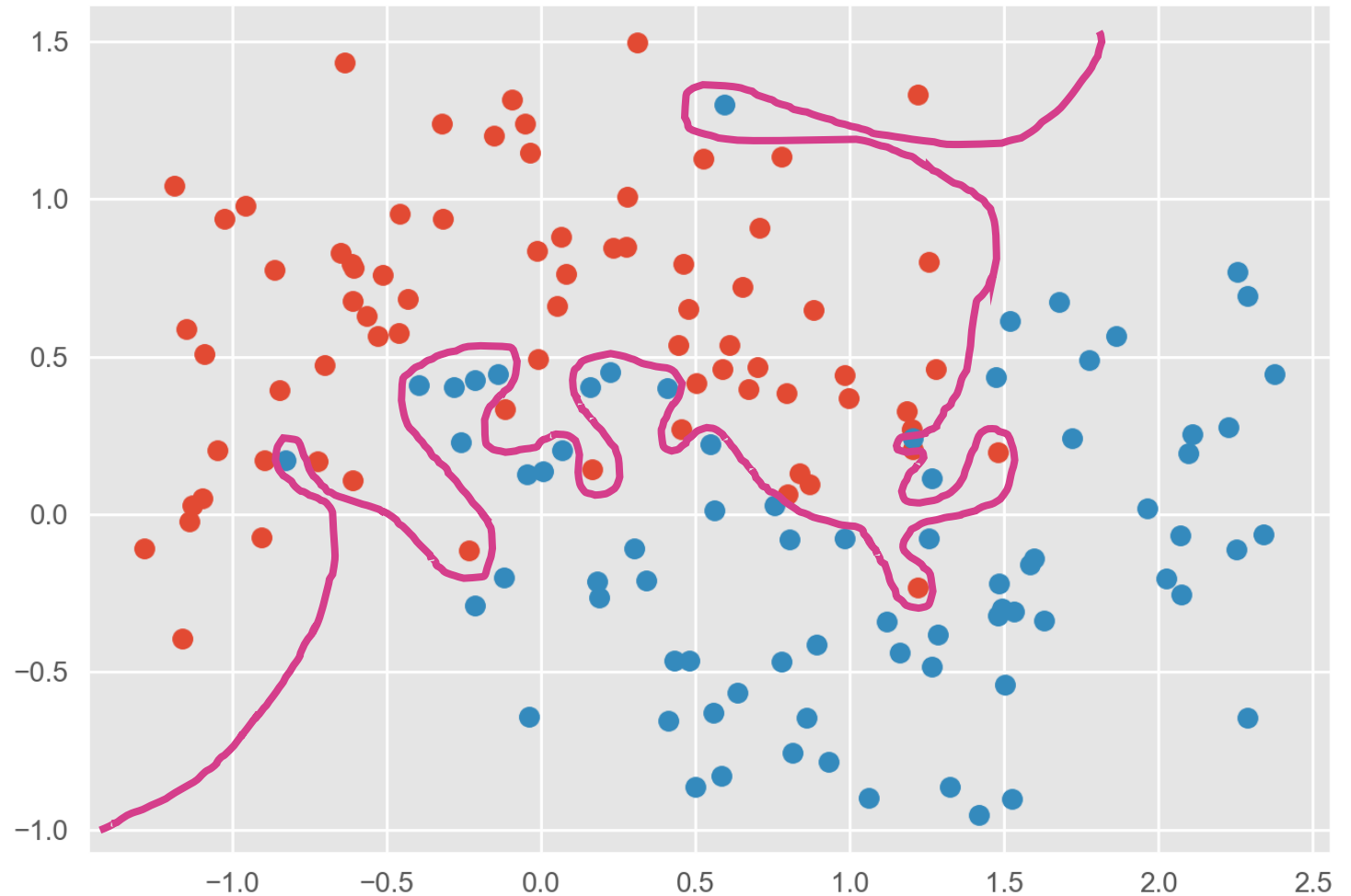
Just like our robot, it doesn't know when to stop.



# We need a strategy to decide when to stop.

Strategy 1:  
*Penalize complexity*  
(regularization)

We could quantify how curvy  
the line is and add that value to  
the loss.

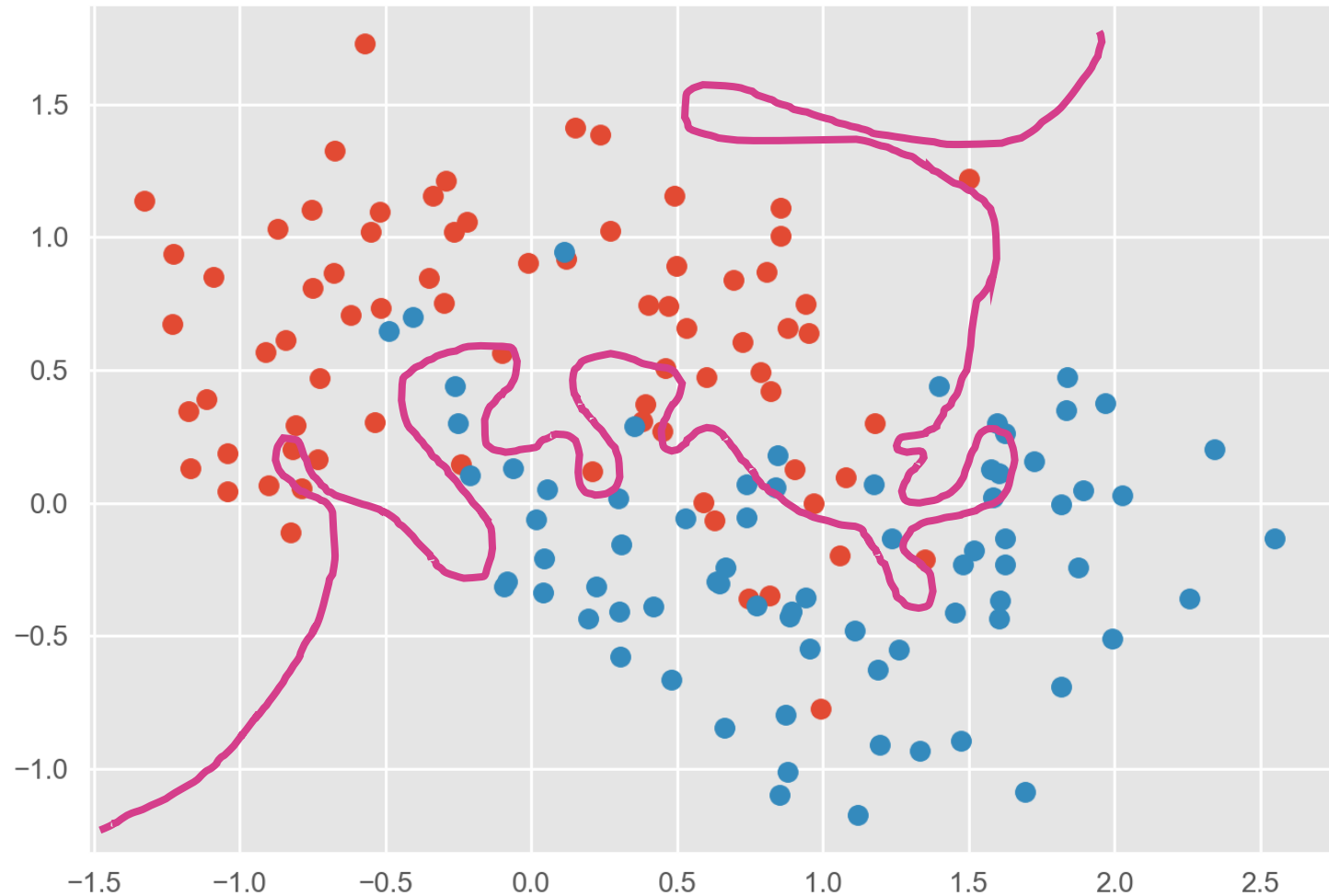


# We need a strategy to decide when to stop.

Strategy 2:

*Keep checking the validation set  
(early stopping)*

When our boundary no longer  
works well on new data, we  
know we've gone too far.



# Strategy 1: Explicit Regularization

- We add a term to the loss ( $L$ ) that quantifies complexity
- By minimizing the loss, we're now striking a balance between two competing objectives:
  1. Improve goodness of fit (i.e. reduce cross-entropy loss  $H$ )
  2. Limit complexity (i.e. reduce regularization term  $R$ )
- The hyperparameter  $\lambda$  controls how much we care about (1) vs (2)

$$L = H + \lambda R$$

# Examples of Explicit Regularization

- L2 (i.e. *ridge*) regularization:  $R = \sum_i \beta_i^2$
- In other words, we penalize the sum of squares of parameters
- Common in linear models and in neural networks

$$L = H + \lambda R$$



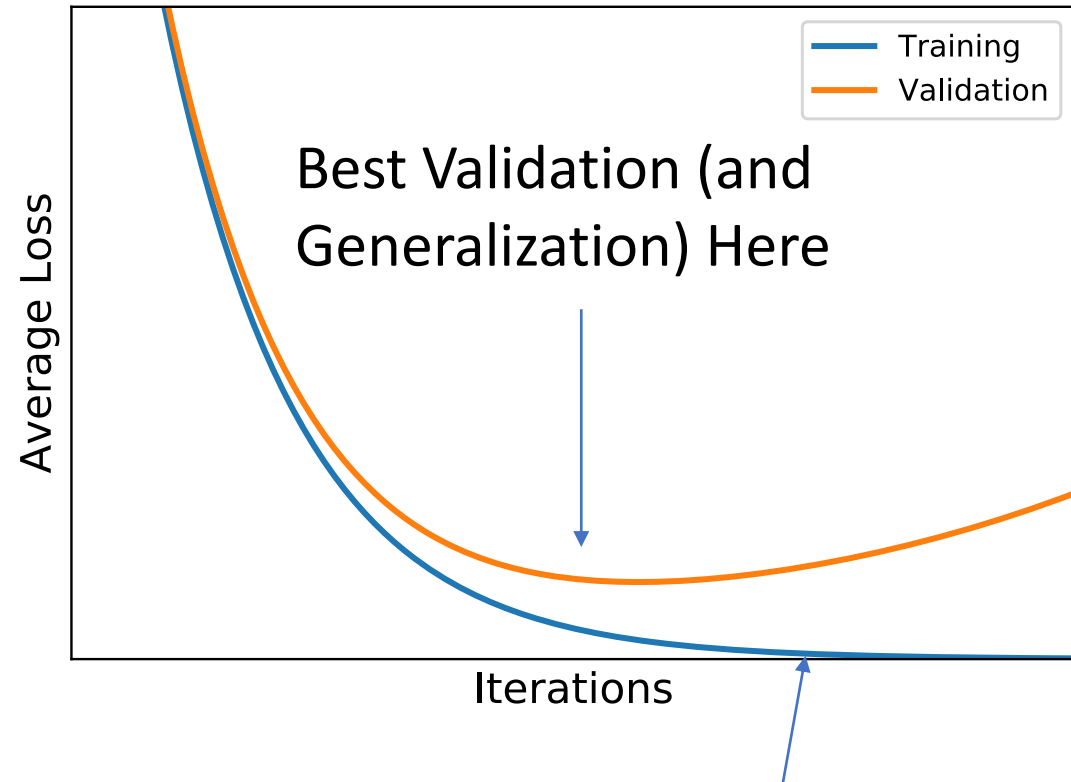
# Examples of Explicit Regularization

- L1 (i.e. LASSO) regularization:  $R = \sum_i |\beta_i|$
- In other words, we penalize the sum of absolute values of parameters
- Common in linear models; less so in neural networks
- Tends to shrink many parameters to 0
  - This may be viewed as a form of feature selection
  - This also improves interpretability

$$L = H + \lambda R$$

# Strategy 2: 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



Training Loss Keeps Improving

# Strategy 3: Dropout

- Exclusive to neural networks
- During training, we set some hidden features to zero at random (different features for each training step)
- This has a dual effect:
  - Parameter shrinkage (similar to L2)
  - Encourages redundancy / redundant features; in other words, the network must model the relationship between  $x$  and  $y$  in multiple ways

# Mitigating Overfitting in Practice

- Machine learning almost always uses some form of regularization and/or early stopping
- Very common to use multiple forms at once
  - Dropout + early stopping in neural networks
  - Combination of L1 and L2 regularization in linear models (i.e. elastic net)
- The validation set is the key: used to choose between strategies
- You will see this when working on data science projects.

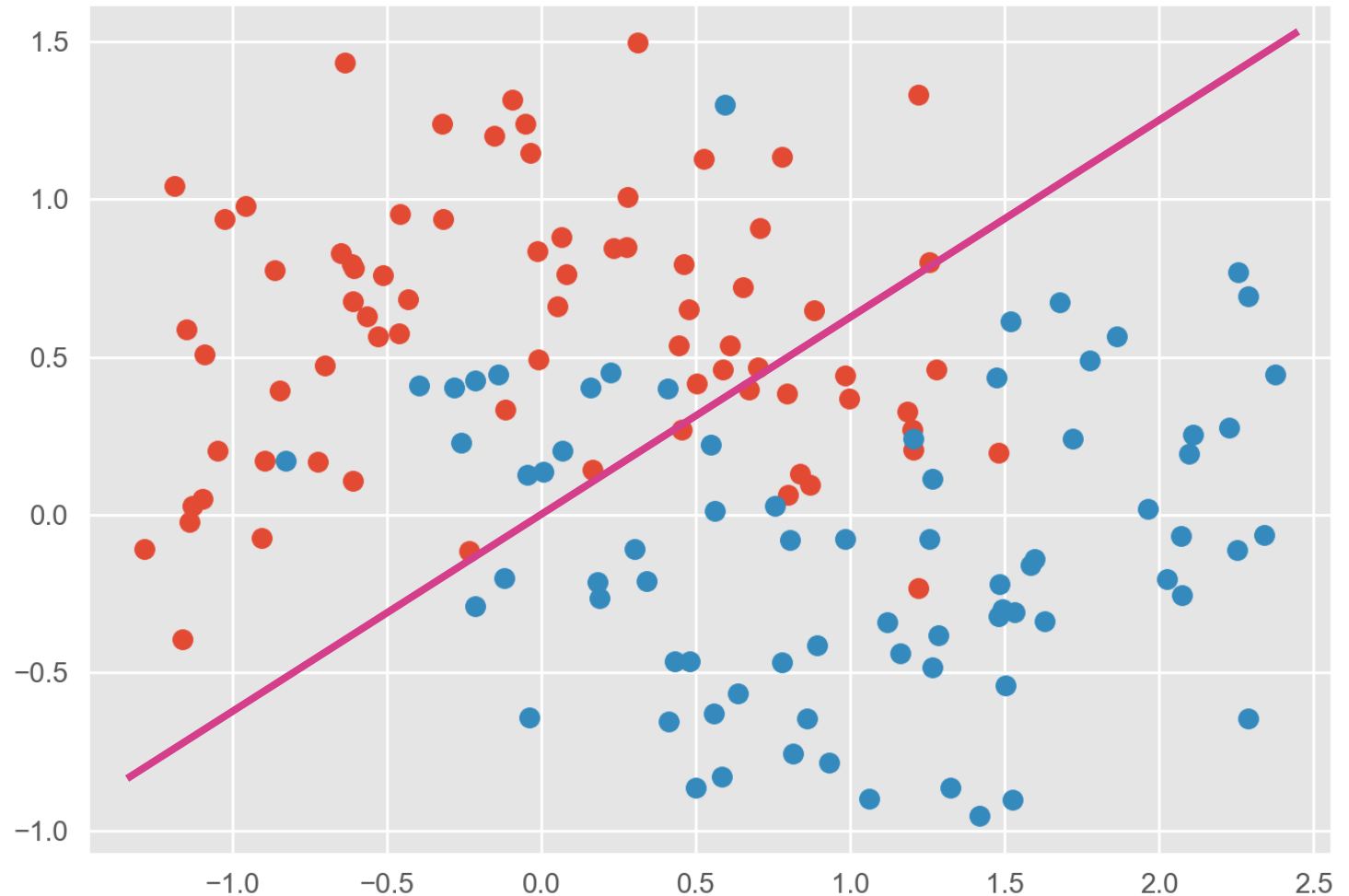
# Linear models can also overfit!!! (How?)

## Linear decision boundary

In this scenario ->  
(# points  $\gg$  # predictors)  
it is not possible.

However, as # points  
approaches # predictors,  
we will see overfitting.

And for # points  $\leq$  # predictors  
(assuming linear independence),  
we can memorize the dataset.





# Scenarios

How to mitigate overfitting?

# Scenario 1: Interpretable CDP

- You are building a clinical decision support system using variables from structured EHR fields
- You have a thousand predictors and a few thousand data points (i.e. patients) in your training set
- You want it to be easy for providers to understand how the model works, and what features are important

## Scenario 2: Pure prediction

- You are building a neural network-based model that identifies nicotine withdrawal from digital health data
- Your input data is a high-dimensional time series of physiologic measurements from multiple sensors
- There is no need to interpret or explain the model: you care only about out-of-sample performance

# Overfitting in linear models

(time permitting)

Suppose we have the following data

Patient	Predictor 1 (numeric)	Predictor 2 (numeric)	Outcome (numeric)
A	.5	.75	1.5
B	1	.75	1.25

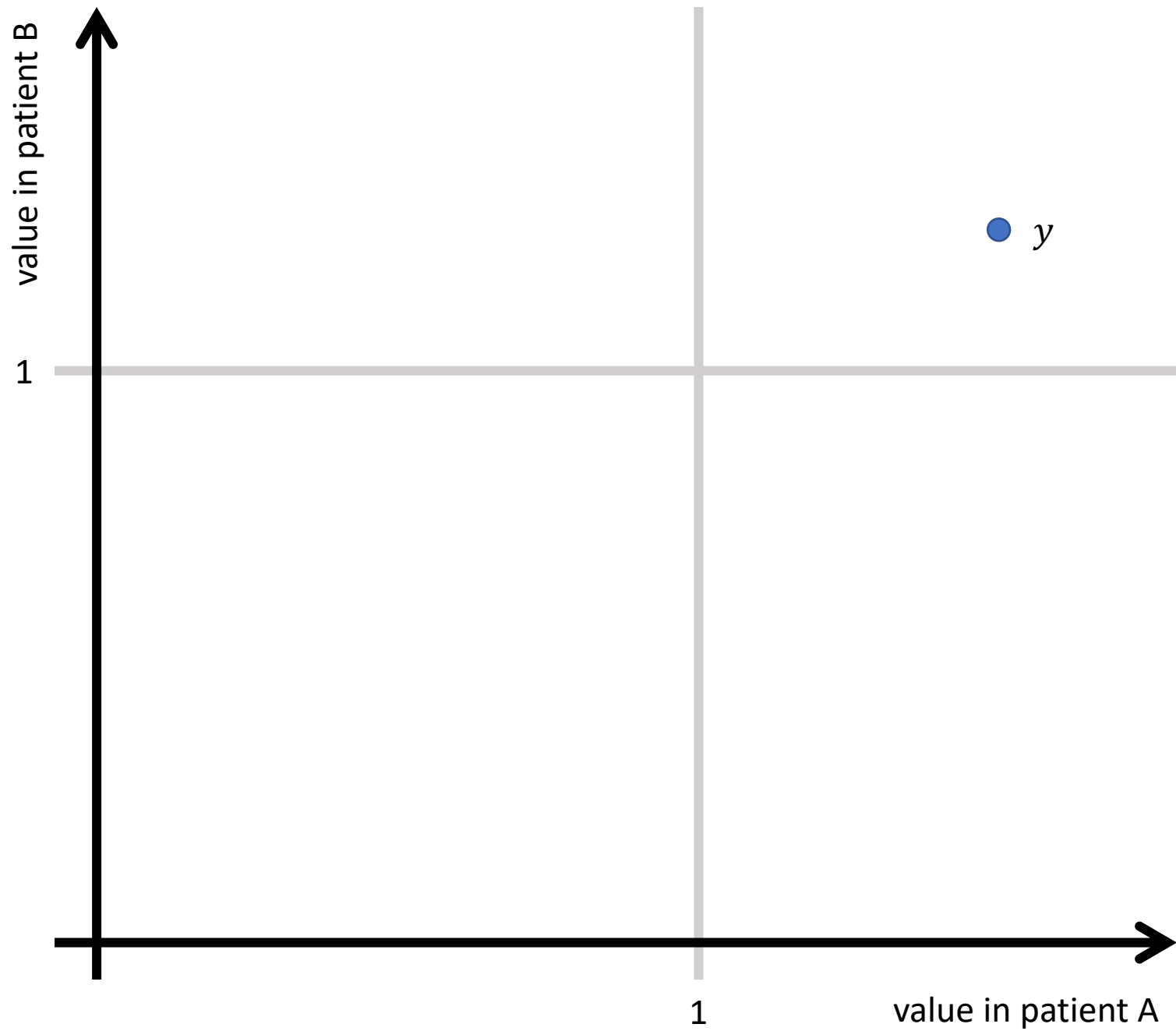
> Goal: find the linear equation that best predicts the outcome

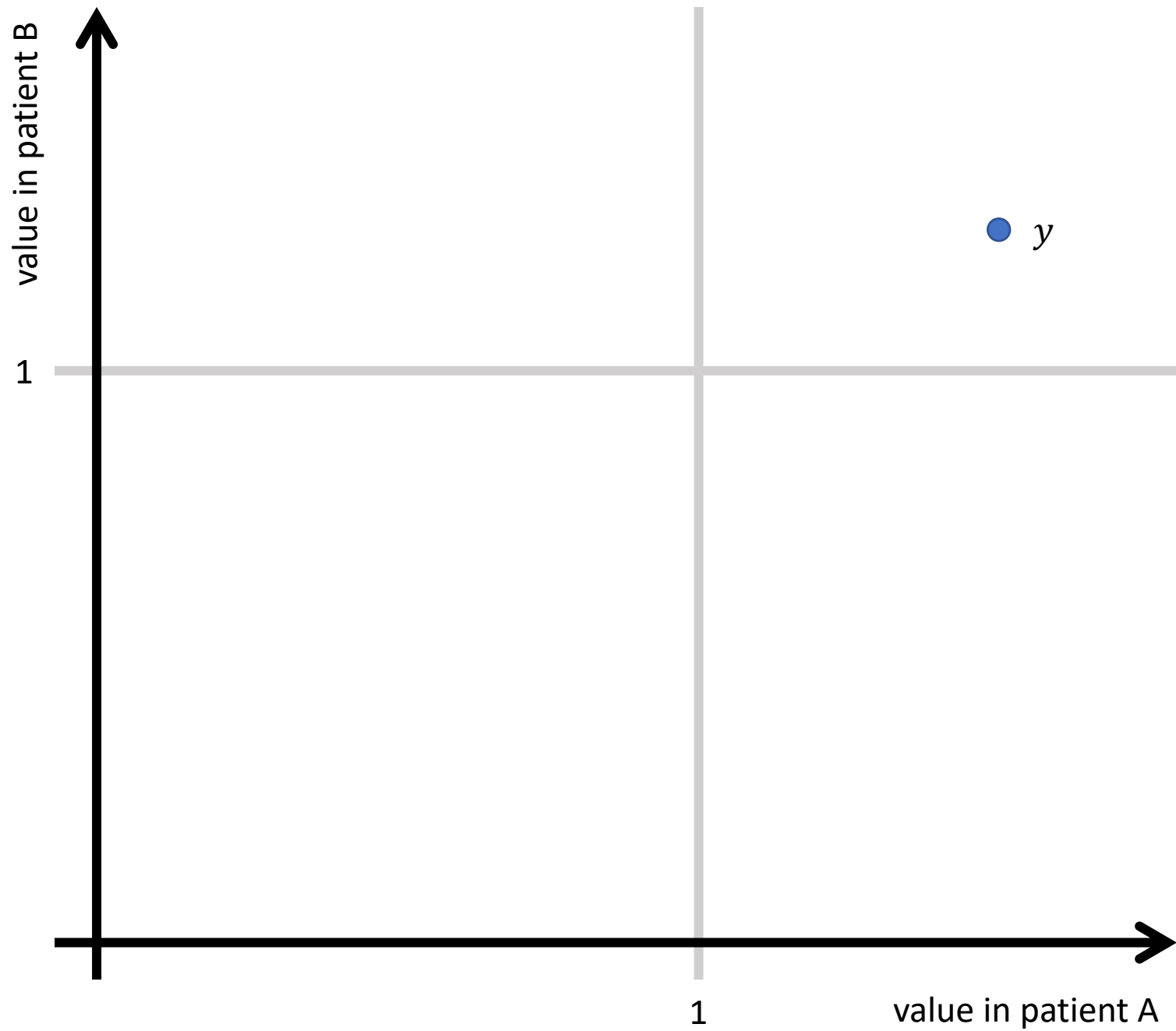


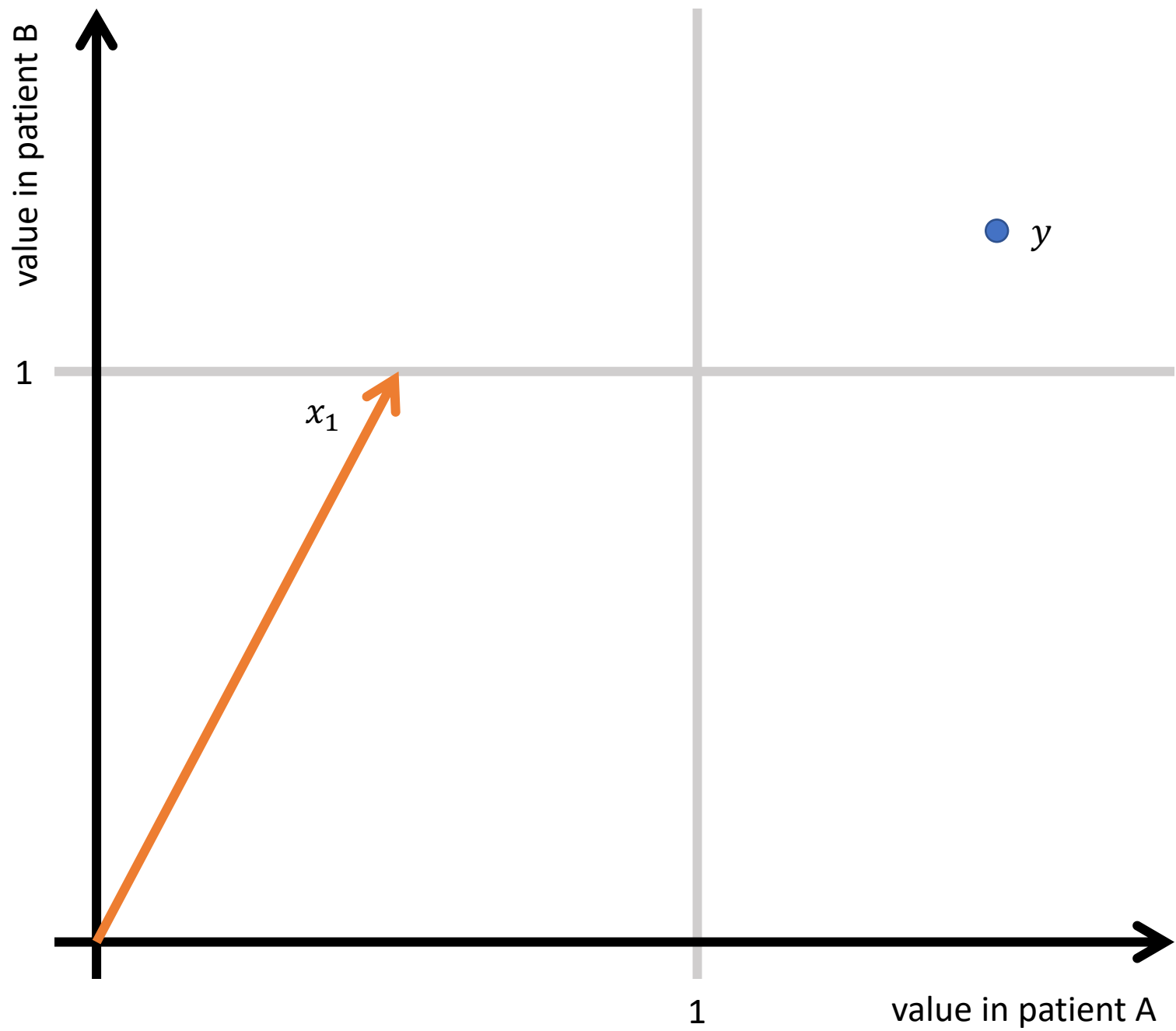
Patient	$x_1$	$x_2$	$y$
A	.5	.75	1.5
B	1	.75	1.25

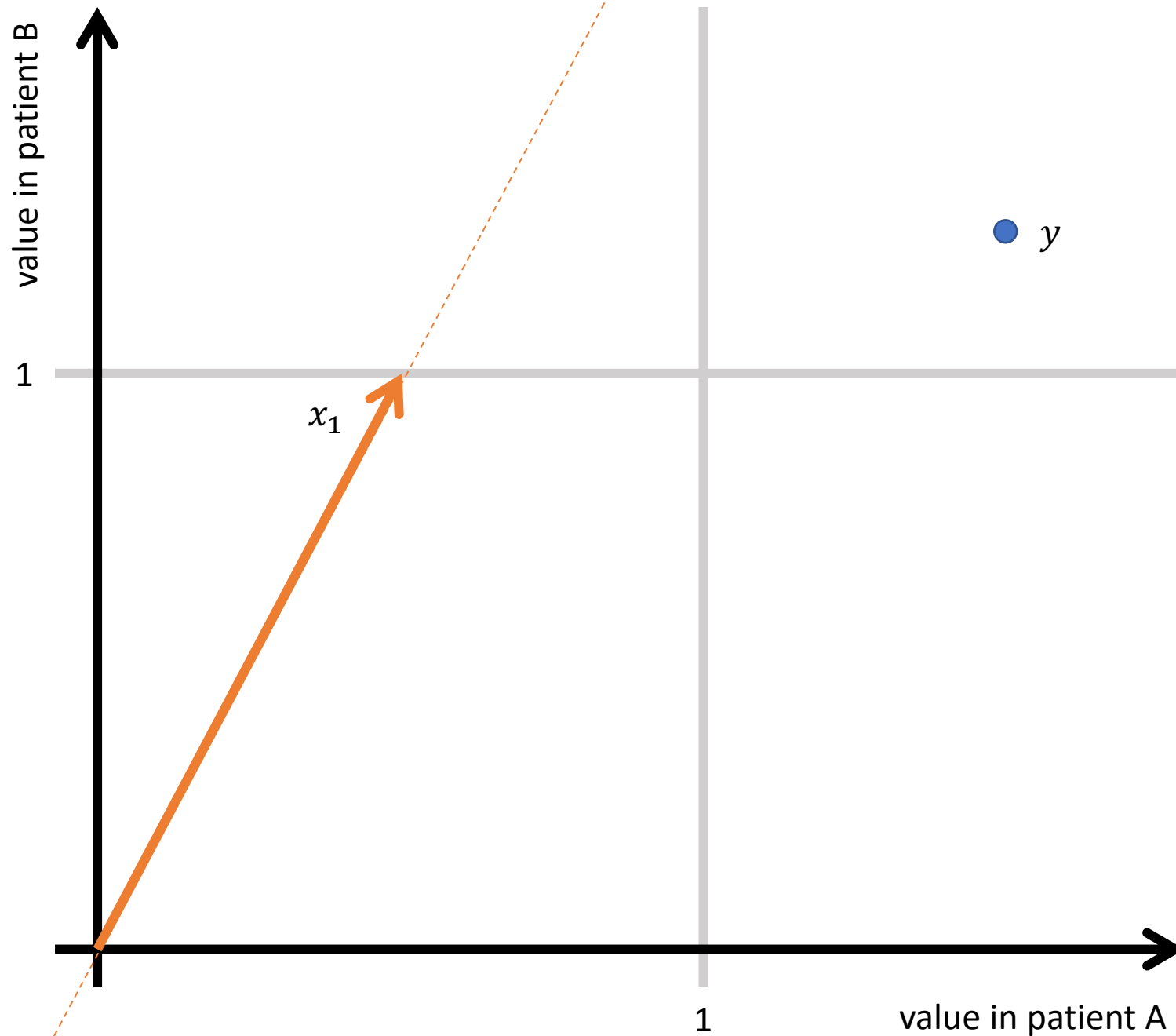
> Goal: find the linear equation that best predicts the outcome

$$b_1x_1 + b_2x_2 = y$$









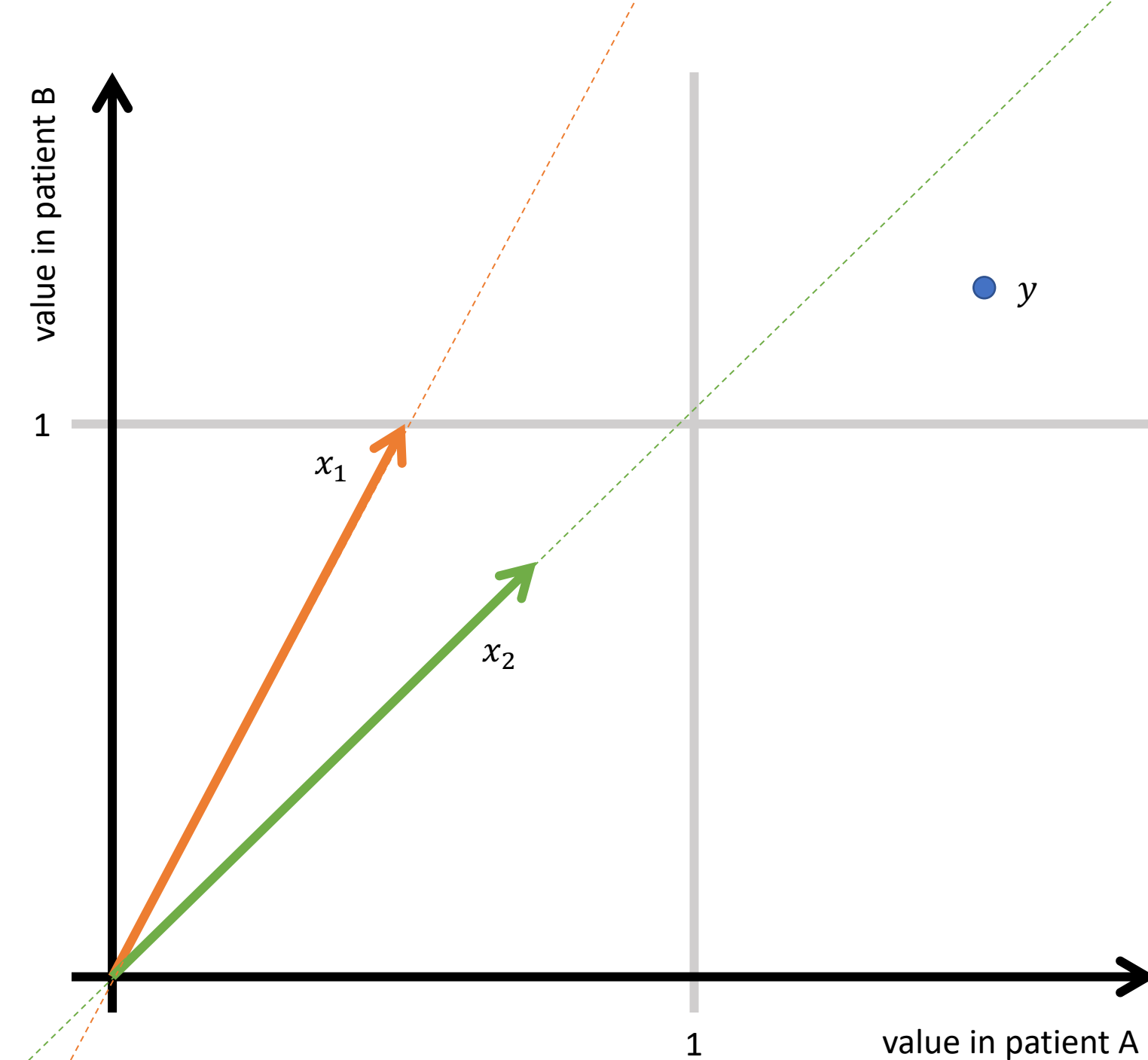
**Goal:** Predict  $y$  with  $x_1$  only

From a graphical perspective, our goal is to get as close as possible to  $y$ , but we can only move in the  $x_1$  direction

-----

$$p < n$$

More patients than predictors.



**Goal:** Predict  $y$  with  $x_1, x_2$

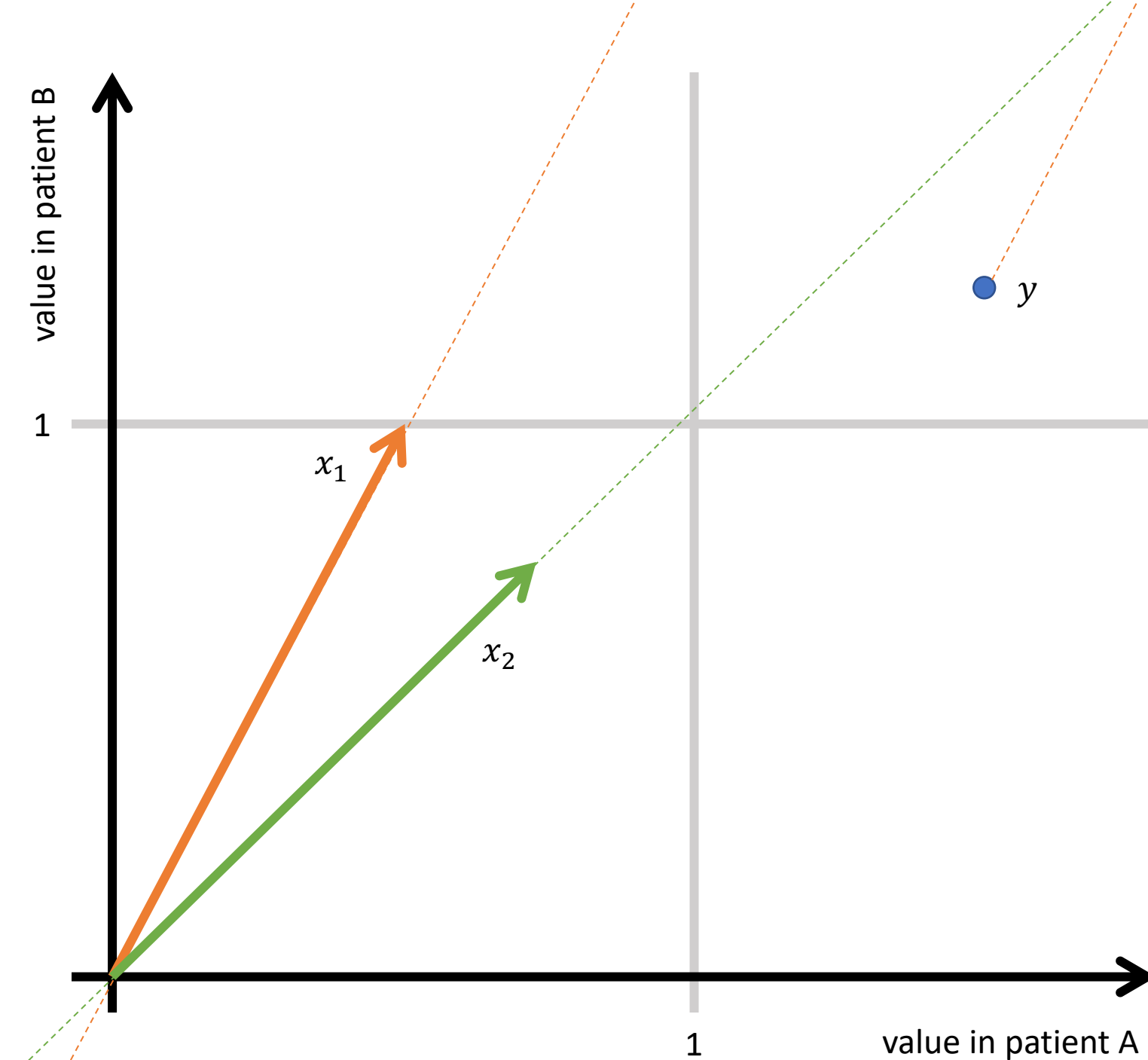
From a graphical perspective, our goal is to get as close as possible to  $y$ . We can now move in both the  $x_1$  direction and the  $x_2$  direction.

---

$$p = n$$

Can always\* predict perfectly on training set

\*assuming linearly independent predictors



**Goal:** Predict  $y$  with  $x_1, x_2$

From a graphical perspective, our goal is to get as close as possible to  $y$ . We can now move in both the  $x_1$  direction and the  $x_2$  direction.

---

$$p = n$$

Can always\* predict perfectly on training set

\*assuming linearly independent predictors

# Takeaways

- When  $\# \text{ points} \leq \# \text{ predictors}$ , we can get perfect predictions (i.e. memorize the dataset)
- However, this typically requires very large model coefficients
- Regularization therefore prevents this behavior, and therefore tends to give us better generalization (i.e. out of sample) performance



# Cross-Validation

(time permitting)

# What if we don't have enough data for this?

- Very common in medicine
- We use *cross validation*

