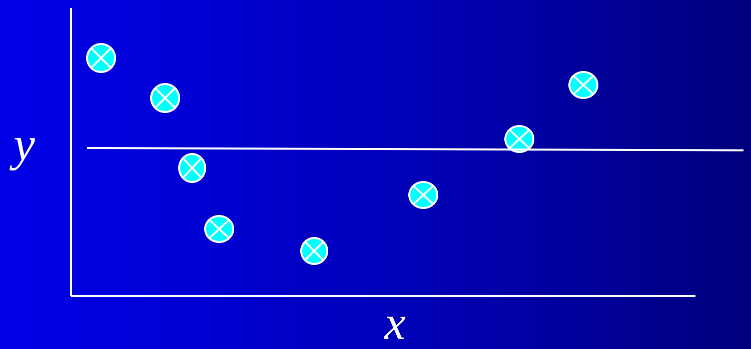
Overfit and Inductive Bias: How to generalize on novel data

Non-Linear Tasks

- Linear Regression will not generalize well to the task below
- Needs a non-linear surface – Could use one of our future models
- Could also do a feature pre-process like with the quadric machine
  - For example, we could use an arbitrary polynomial in *x*
  - Thus, it is still linear in the coefficients, and can be solved with delta rule

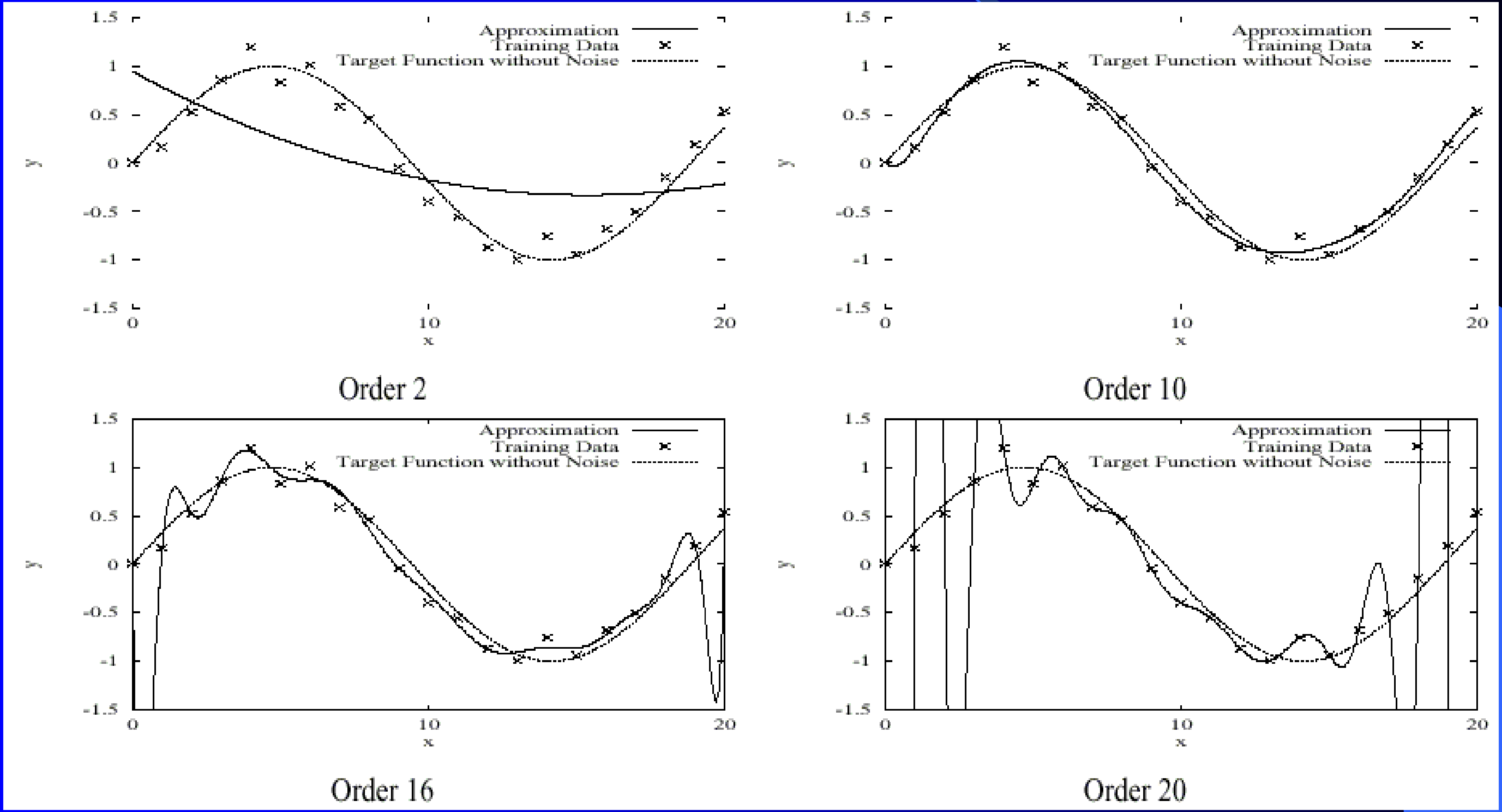
$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_n X^n$$

  - What order polynomial should we use? – Overfit issues can occur



Overfitting

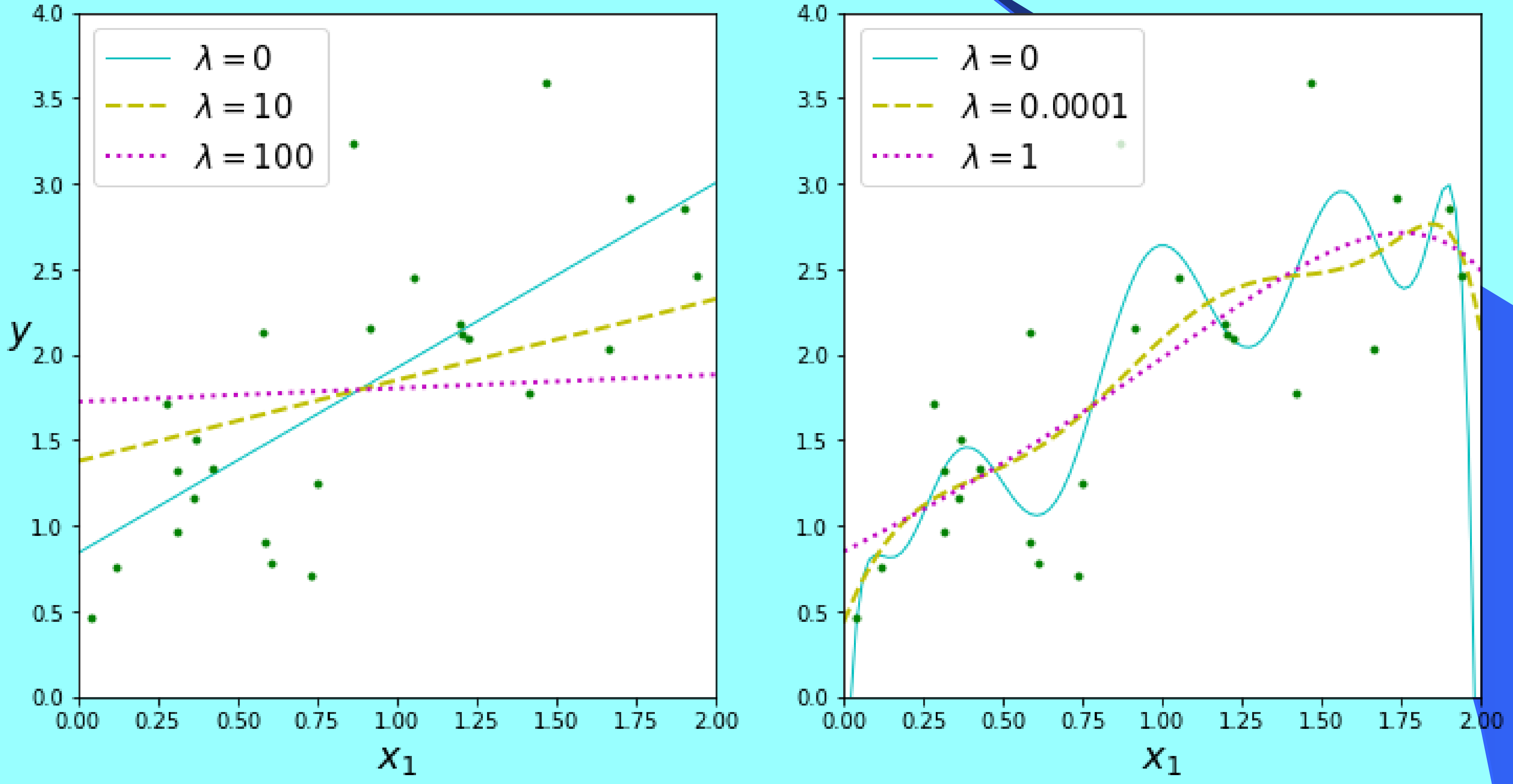
Typically try to learn a model just complex enough to do well and no more complex than that



Linear Regression Regularization

- Keep the model as simple as possible while still being accurate
  - For regression, keep the function smooth
- Regularization approach: updated loss function
  - Minimize $F(h) = Error(h) + \lambda \cdot Complexity(h)$
  - Tradeoff training accuracy vs complexity
- Ridge Regression (L2 regularization) – Minimize:
  - $F(\mathbf{w}) = TSS(\mathbf{w}) + \lambda||\mathbf{w}||^2 = \Sigma\,(predicted_i - actual_i)^2 + \lambda\Sigma w_i^2$
  - Gradient of $F(\mathbf{w})$: $\boxed{\Delta w_i = c(t - net)x_i - \lambda w_i}$ (Weight decay, not bias weight)
  - Linear regression with just the original features cannot overfit, but regularization useful when the features are a non-linear transform of the initial features (e.g. polynomials in *x*)
  - Lasso regression uses an L1 vs an L2 weight penalty: $\lambda\Sigma|w_i|$ and thus decay is just λ since derivative drops weight from the term
  - Sometimes called shrinkage models – decrease/shrink weights

Ridge Regression Example



Hypothesis Space

- The Hypothesis space *H* is the set of all possible models *h* which can be learned by the current learning algorithm
  - e.g. Set of possible weight settings for a perceptron
- Restricted hypothesis space
  - Can be easier to search
  - May avoid overfit since they are usually simpler (e.g. linear or low order decision surface)
  - Often will underfit – e.g linear models
- Unrestricted Hypothesis Space
  - Can represent any possible function and thus can fit the training set well – Includes most of the powerful ML algorithms
  - However, mechanisms must be used to avoid overfit

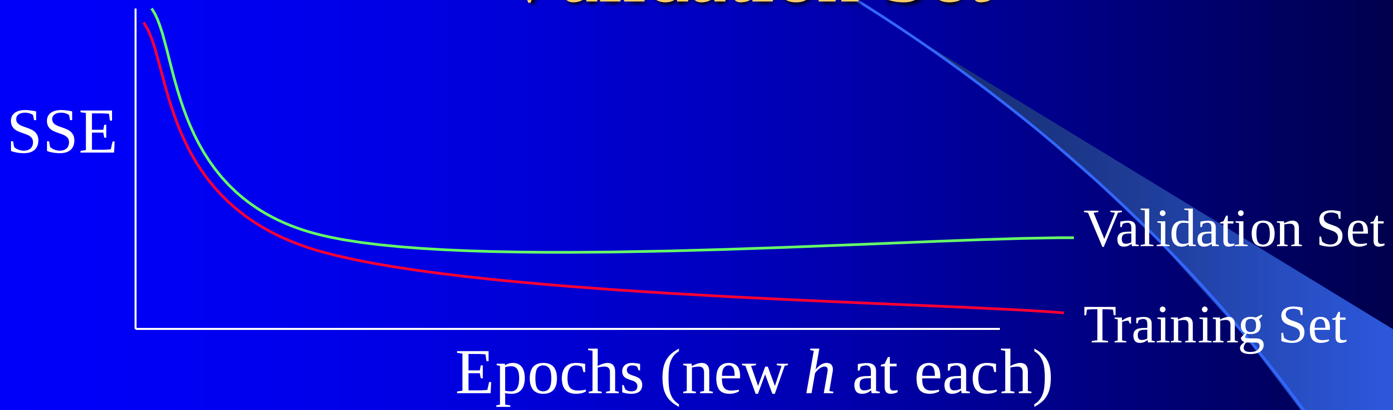
Avoiding Overfit

- Regularization: *any modification we make to learning algorithm that is intended to reduce its generalization error but not its training error*
- Occam's Razor – William of Ockham (c. 1287-1347)
  - Favor simplest explanation which fits the data
- General Key: Focus on patterns/rules that really matter and ignore others
- Simplest accurate model: accuracy vs. complexity trade-off. Find $h \in H$ which minimizes an objective function of the form:

$$F(h) = Error(h) + \lambda \cdot Complexity(h)$$

  - Complexity could be number of nodes, size of tree, magnitude of weights, etc.
- More Training Data (vs. overtraining on same data)
  - Data set augmentation – Fake data, Can be very effective, Jitter, but take care…
  - Denoising – add random noise to inputs during training – can act as a regularizer
  - Adding noise to models. e.g. (Random Forests, Dropout , discuss with ensembles)
- *Early Stopping* – Very common regularization approach: Start with simple model (small parameters/weights) and stop training as soon as we attain good generalization accuracy (and before parameters get large)
  - Common early stopping approach is to use a validation set (next slide)
- We will discuss other model specific approaches with specific models

Early Stopping/Model Selection with a Validation Set



- There is a different model *h* after each epoch
- Select a model in the area where the validation set accuracy flattens
- Keep *bssf* (Best Solution So Far). Once you go *w* epochs with no improvement stop and use the parameters at the *bssf* *w* epochs ago.
- The validation set comes out of training set data
- Still need a separate test set to use after selecting model *h* to predict future accuracy
- Simple and unobtrusive, does not change objective function, etc
  - Can be done in parallel on a separate processor
  - Can be used alone or in conjunction with other regularization approaches

Inductive Bias

- The approach used to decide how to generalize novel cases
- A common approach is Occam's Razor – The *simplest* hypothesis which *explains/fits* the data is usually the best
- Many other rationale biases and variations

** Inductive Bias – Challenge Question **

- The approach used to decide how to generalize novel cases
- A common approach is Occam's Razor – The *simplest* hypothesis which *explains/fits* the data is usually the best
- Many other rationale biases and variations
- All the variables in the shown data set are binary
- A = True, $\bar{A}$ = False
- You be the machine learner, learn the data set, and decide your inductive bias

- You then get the new input $\bar{A}\ B\ C$. What is your generalized output?
  - *A.* Z is false
  - *B.* Z is true
  - C. I can't decide

ABC Þ Z

$A\bar{B}C$ Þ Z

$AB\bar{C}$ Þ Z

$A\bar{B}\bar{C}$ Þ Z

$\bar{A}\bar{B}\bar{C}$ Þ $\bar{Z}$

$\bar{A}BC$ Þ ?

One Definition for Inductive Bias

Inductive Bias: Any basis for choosing one generalization over another, other than strict consistency with the observed training instances

Sometimes just called the *Bias* of the algorithm (don't confuse with the bias weight of a neural network).

Bias-Variance Trade-off – Will discuss in more detail when we discuss ensembles

Inductive Bias Approaches

- Restricted Hypothesis Space - Can just try to minimize error since hypotheses are already simple
  - Linear or low order threshold function
  - k-DNF, k-CNF, etc.
  - Low order polynomial
- Preference Bias – Use unrestricted hypothesis space, but “prefer” one hypothesis over another even though they have similar training accuracy
  - Occam’s Razor
  - “Smallest” DNF representation which matches well
  - Shallow decision tree with high information gain
  - Neural Network with low validation error and small magnitude weights

Need for Bias

2^{2^n} Boolean functions of n inputs

x1	x2	x3	Class	Possible Consistent Function Hypotheses
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	1	
1	0	0		
1	0	1		
1	1	0		
1	1	1	?	

Need for Bias

2^{2^n} Boolean functions of *n* inputs

x1	x2	x3	Class	Possible Consistent Function Hypotheses
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0		0
1	0	1		0
1	1	0		0
1	1	1	?	0

Need for Bias

2^{2^n} Boolean functions of n inputs

x1	x2	x3	Class	Possible Consistent Function Hypotheses	
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0		0	0
1	0	1		0	0
1	1	0		0	0
1	1	1	?	0	1

Need for Bias

2^{2^n} Boolean functions of *n* inputs

x1	x2	x3	Class	Possible Consistent Function Hypotheses															
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0		0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	1		0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0		0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	1	?	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Without an Inductive Bias we have no rationale to choose one hypothesis over another and thus a random guess would be as good as any other option.

Need for Bias

2^{2^n} Boolean functions of n inputs

x1	x2	x3	Class	Possible Consistent Function Hypotheses															
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0		0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	1		0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0		0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	1	?	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Inductive Bias guides which hypothesis we should prefer?
What happens in this case if we use simplicity (Occam's Razor) as our inductive Bias (preference bias)?

Learnable Problems

- "Raster Screen" Problem
- Pattern Theory
  - Regularity in a task
  - Compressibility
- Don't care features and Impossible states
- Interesting/Learnable Problems
  - What we actually deal with
  - Can we formally characterize them?
- Learning a training set vs. generalizing
  - A function where each output is set randomly (coin-flip)
  - Output class is independent of all other instances in the data set
- Computability vs. Learnability (Optional)

Computable and Learnable Functions

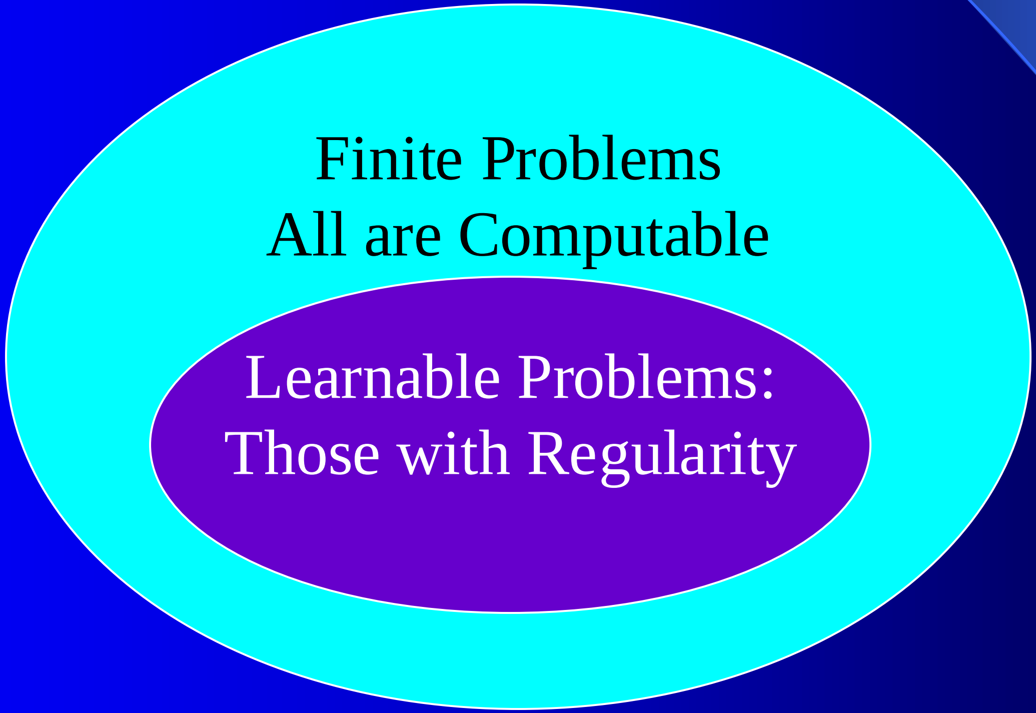
- Can represent any function with a look-up table (Addition)
  - Finite function/table – Fixed/capped input size
  - Infinite function/table – arbitrary finite input size
  - All finite functions are computable – Why?
  - Infinite addition computable because it has regularity which allows us to represent the infinite table with a finite representation/program
- Random function – outputs are set randomly
  - Can we compute these?
  - Can we learn these?
    - Assume learnability means we can get better than random when classifying novel examples
- Arbitrary functions – Which are computable?
- Arbitrary functions – Which are learnable?

Computability and Learnability – Finite Problems

- Finite problems assume finite number of mappings (Finite Table)
  - Fixed input size arithmetic
  - Random memory in a RAM
- Learnable: Can do better than random on novel examples

Computability and Learnability – Finite Problems

- Finite problems assume finite number of mappings (Finite Table)
  - Fixed input size arithmetic
  - Random memory in a RAM
- Learnable: Can do better than random on novel examples

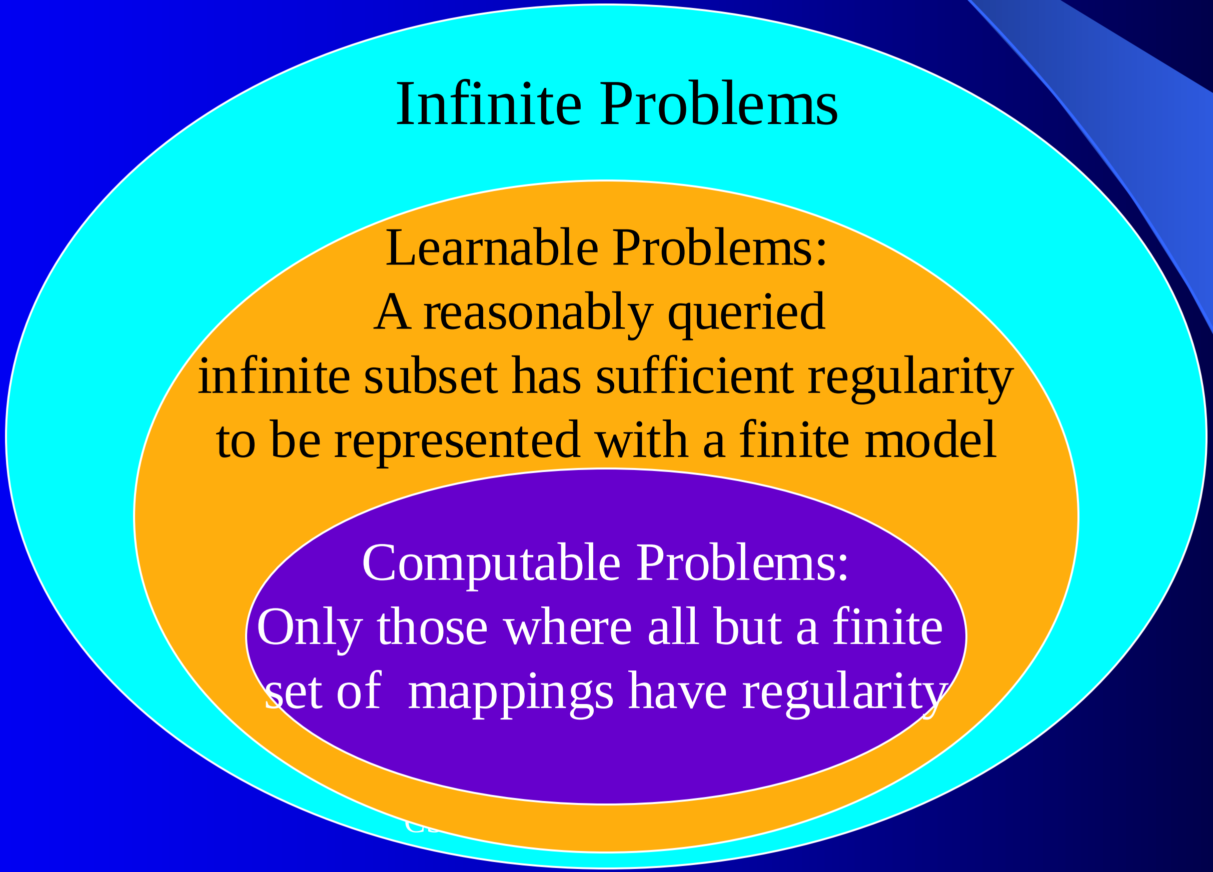


Computability and Learnability – Infinite Problems

- Infinite number of mappings (Infinite Table)
  - Arbitrary input size arithmetic
  - Halting Problem (no limit on input size)
  - Do two arbitrary strings match

Computability and Learnability – Infinite Problems

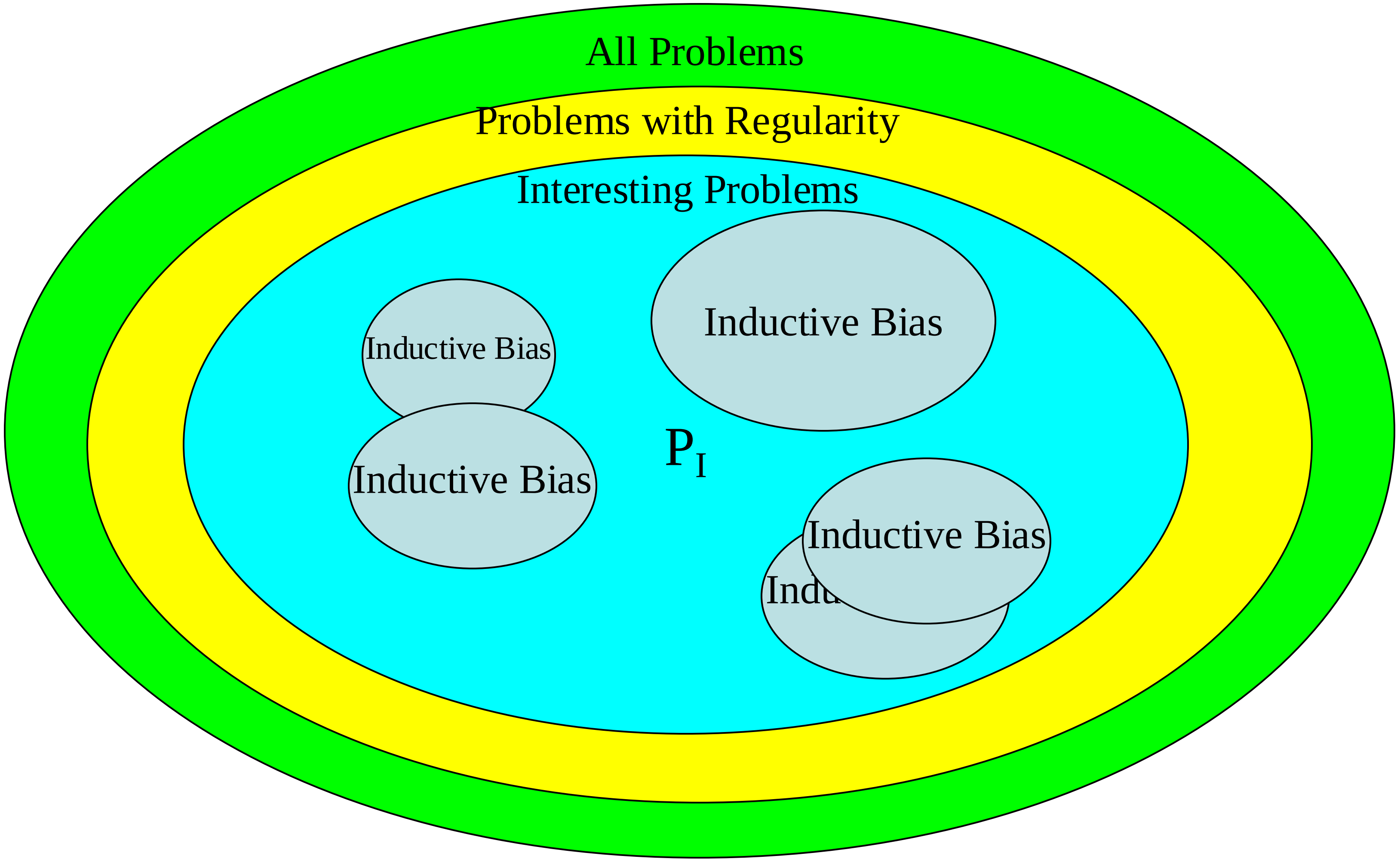
- Infinite number of mappings (Infinite Table)
  - Arbitrary input size arithmetic
  - Halting Problem (no limit on input size)
  - Do two arbitrary strings match


Infinite Problems
Learnable Problems:
A reasonably queried
infinite subset has sufficient regularity
to be represented with a finite model
Computable Problems:
Only those where all but a finite
set of mappings have regularity

No Free Lunch

- Any inductive bias chosen will have equal accuracy compared to any other bias over *all* possible functions/tasks, assuming all functions are equally likely. If a bias is correct on some cases, it must be incorrect on equally many cases.
- Is this a problem?
  - Random vs. Regular
  - Anti-Bias? (even though regular)
  - The “Interesting” Problems – subset of learnable?
- Are all functions equally likely in the real world?

Interesting Problems and Biases



More on Inductive Bias

- Inductive Bias requires some set of prior assumptions about the tasks being considered and the learning approaches available
- Tom Mitchell's definition: Inductive Bias of a learner is the set of additional assumptions sufficient to justify its inductive inferences as deductive inferences
- We consider standard ML algorithms/hypothesis spaces to be different inductive biases: C4.5 (Greedy best attributes), Backpropagation (simple to complex), etc.

Which Bias is Best?

- Not one Bias that is best on all problems
- Our experiments
  - Over 50 real world problems
  - Over 400 inductive biases – mostly variations on critical variable biases vs. similarity biases
- Different biases were a better fit for different problems
- Given a data set, which Learning model (Inductive Bias) should be chosen?

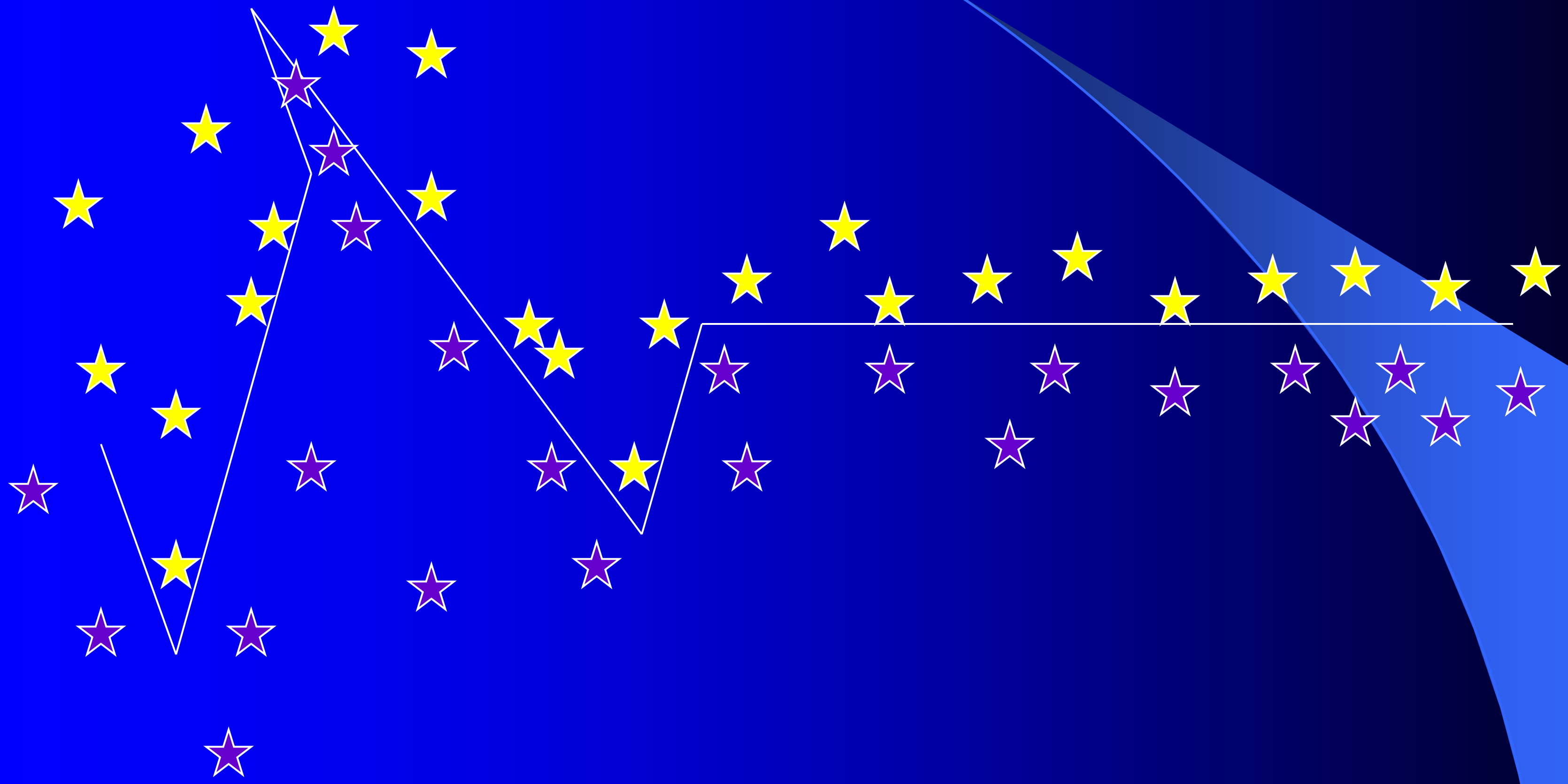
Automatic Discovery of Inductive Bias

- Defining and characterizing the set of Interesting/Learnable problems
- To what extent do current biases cover the set of interesting problems
- Automatic feature selection
- Automatic selection of Bias (before and/or during learning), including all learning parameters
- Dynamic Inductive Biases (in time and space)
- Combinations of Biases – Ensembles, Oracle Learning

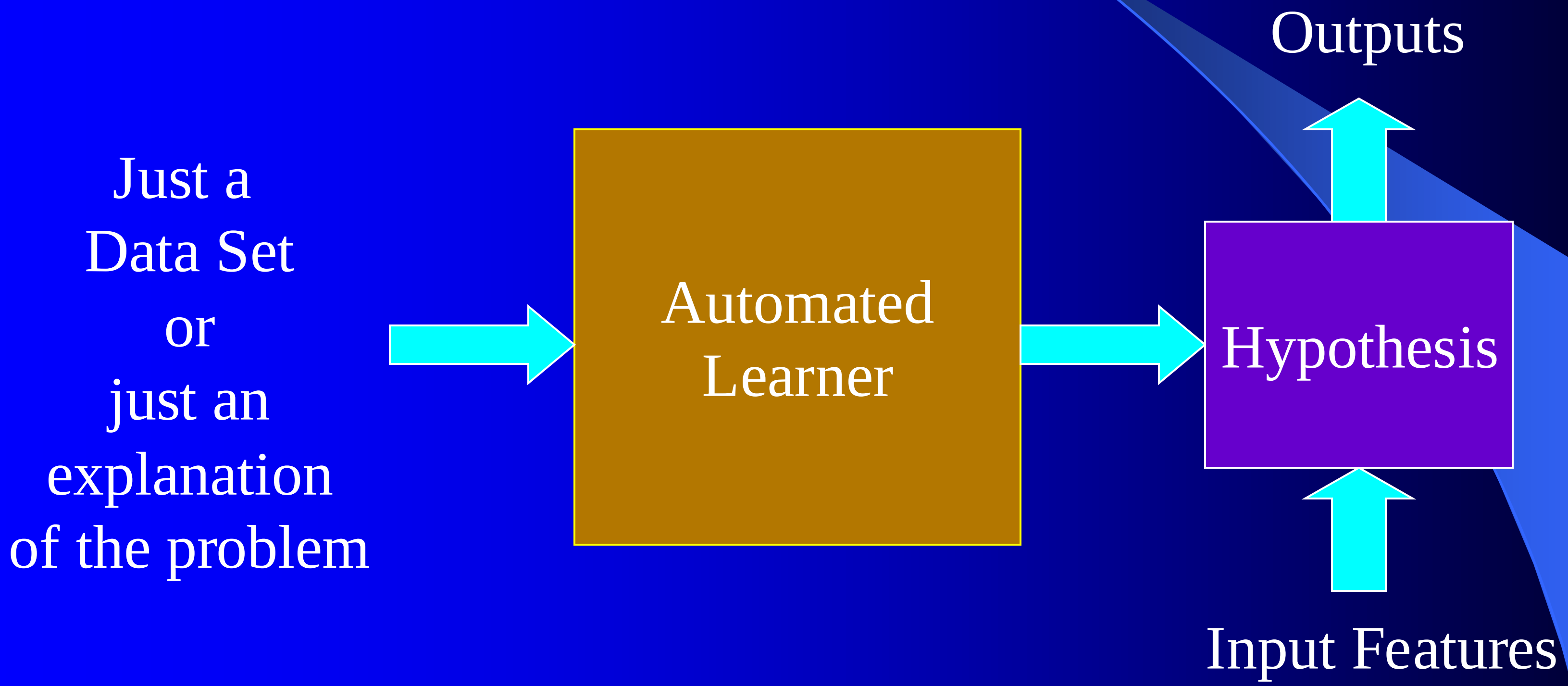
Dynamic Inductive Bias in Time

- Can be discovered as you learn
- May want to learn general rules first followed by true exceptions
- Can be based on ease of learning the problem
- Example: SoftProp – From Lazy Learning to Backprop

Dynamic Inductive Bias in Space



ML Holy Grail: We want all aspects of the learning mechanism automated, including the Inductive Bias



BYU Neural Network and Machine Learning Laboratory Work on Automatic Discover of Inductive Bias

- Proposing New Learning Algorithms (Inductive Biases)
- Theoretical issues
  - Defining the set of Interesting/Learnable problems
  - Analytical/empirical studies of differences between biases
- Ensembles – Wagging, Mimicking, Oracle Learning, etc.
- Meta-Learning – A priori decision regarding which learning model to use
  - Features of the data set/application
  - Learning from model experience
- Automatic selection of Parameters
  - Constructive Algorithms – ASOCS, DMPx, etc.
  - Learning Parameters – Windowed momentum, Automatic improved distance functions (IVDM)
- Automatic Bias in time – SoftProp
- Automatic Bias in space – Overfitting, sensitivity to complex portions of the space: DMP, higher order features

Your Project Proposals

- Examples – Look at Irvine Data Set to get a feel of what data sets look like
- Stick with supervised classification problems for the most part for the project proposals
- Tasks which interest you
- Too hard vs Too Easy
  - Data should be able to be gathered in a relatively short time
  - And, want you to have to battle with the data/features a bit
- See description in Learning Suite
  - Remember your example instance!

Feature Selection, Preparation, and Reduction

- Learning accuracy depends on the data!
  - *Is the data representative of future novel cases* - critical
  - Relevance
  - Amount
  - Quality
    - Noise
    - Missing Data
    - Skew
  - Proper Representation
  - How much of the data is labeled (output target) vs. unlabeled
  - Is the number of features/dimensions reasonable?
    - Reduction

Gathering Data

- Consider the task – What kinds of features could help
- Data availability
  - Significant diversity in cost of gathering different features
  - More the better (in terms of number of instances, not necessarily in terms of number of dimensions/features)
    - The more features you have the more data you need
  - Data augmentation, Jitter – Increased data can help with overfit – handle with care!
- Labeled data is best
- If not labeled
  - Could set up studies/experts to obtain labeled data
  - Use unsupervised and semi-supervised techniques
    - Clustering
    - Active Learning, Bootstrapping, Oracle Learning, etc.

Feature Selection - Examples

- Invariant Data
  - For character recognition: Size, Rotation, Translation Invariance
    - Especially important for visual tasks
  - Chess board features
    - Is vector of board state invariant?
- Character Recognition Class Assignment Example
  - Assume we want to draw a character with an electronic pen and have the system output which character it is
  - What features should we use and how would we train/test the system?

When to Gather More Data

- When trying to improve performance, you may need
  - More Data
  - Better Input Features
  - Different Machine learning models or hyperparameters
  - Etc.
- One way to decide if you need more/better data
  - Compare your accuracy on training and test set
  - If bad training set accuracy then you probably need better data, features, noise handling, etc., or you might need a different learning model/hyperparameters
  - If test set accuracy is much worse than training set accuracy then gathering more data is usually a good direction, though overfit or learning model/hyperparameters could still be a major issue