Course 3, Module 2
Constructing Features for Prediction

CMPUT 365
Fall 2021

Announcements

- Marking of the midterm will be done this week
  - grades will be released via eclass
  - an important policy: **no arguing for marks**
  - If you want to understand the answers, then come to my office hours
- Mini-essay #2 topics are available here:
  - https://docs.google.com/document/d/1EOC5TQh-SKC5zYXrvvC44hR8wthwttrROKwyDPlUBz0

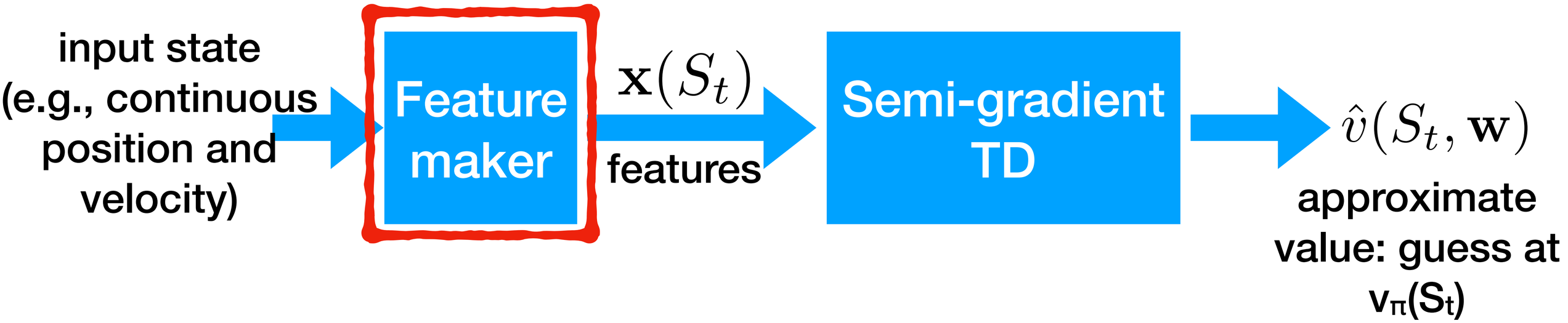
Review of Course 3, Module 2
How to generate features, including Neural Nets

How to make an RL agent: tabular setting

- Pick an algorithm that is a good match for your problem (perhaps you choose Sarsa because it's an episodic cost-to-goal grid world task)
- Decide an exploration method (e.g., e-greedy or optimistic initial values)
- Decide how to initialize the value function (setting Q_0)
- Decide how to set the other parameters (alpha, epsilon, etc)
- Decide how to run the experiment: number of episodes, steps per episodes, what to plot

How to make an RL agent: function approximation setting

- Pick an algorithm that is a good match for your problem
- Decide on linear or non-linear function approximation
- Decide on the function approximator (tile coding, neural network)
  - AND the parameters of the function approximator.
  - **This impacts all the following choices …**
    - Decide on an exploration method (e.g., e-greedy)
    - Decide how to initialize the value function (setting Q_0)
    - Decide how to set the other parameters (alpha, epsilon, etc)
    - Decide how to run the experiment: number of episodes, steps per episodes, what to plot



- What does it mean for $\hat{v}(S_t, \mathbf{w})$ to be a linear function?
  - $\hat{v}(S_t, \mathbf{w}) \doteq \sum_i x_i(S_t) w_i$
  - $x_i(S_t)$ is the i-th component of the feature vector **x_t:**
    - **where does x_t come from?**

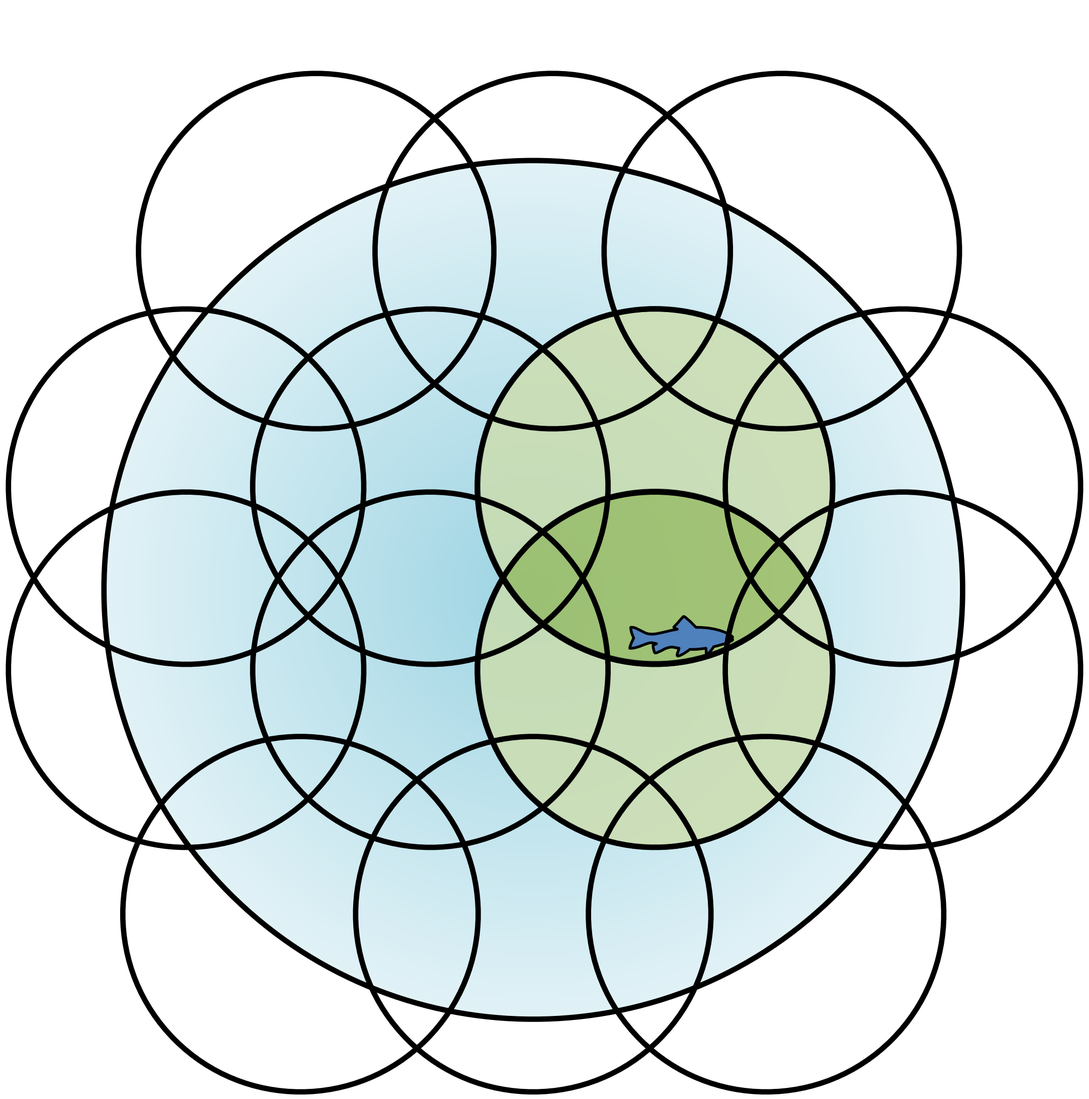
Video 1: Coarse Coding

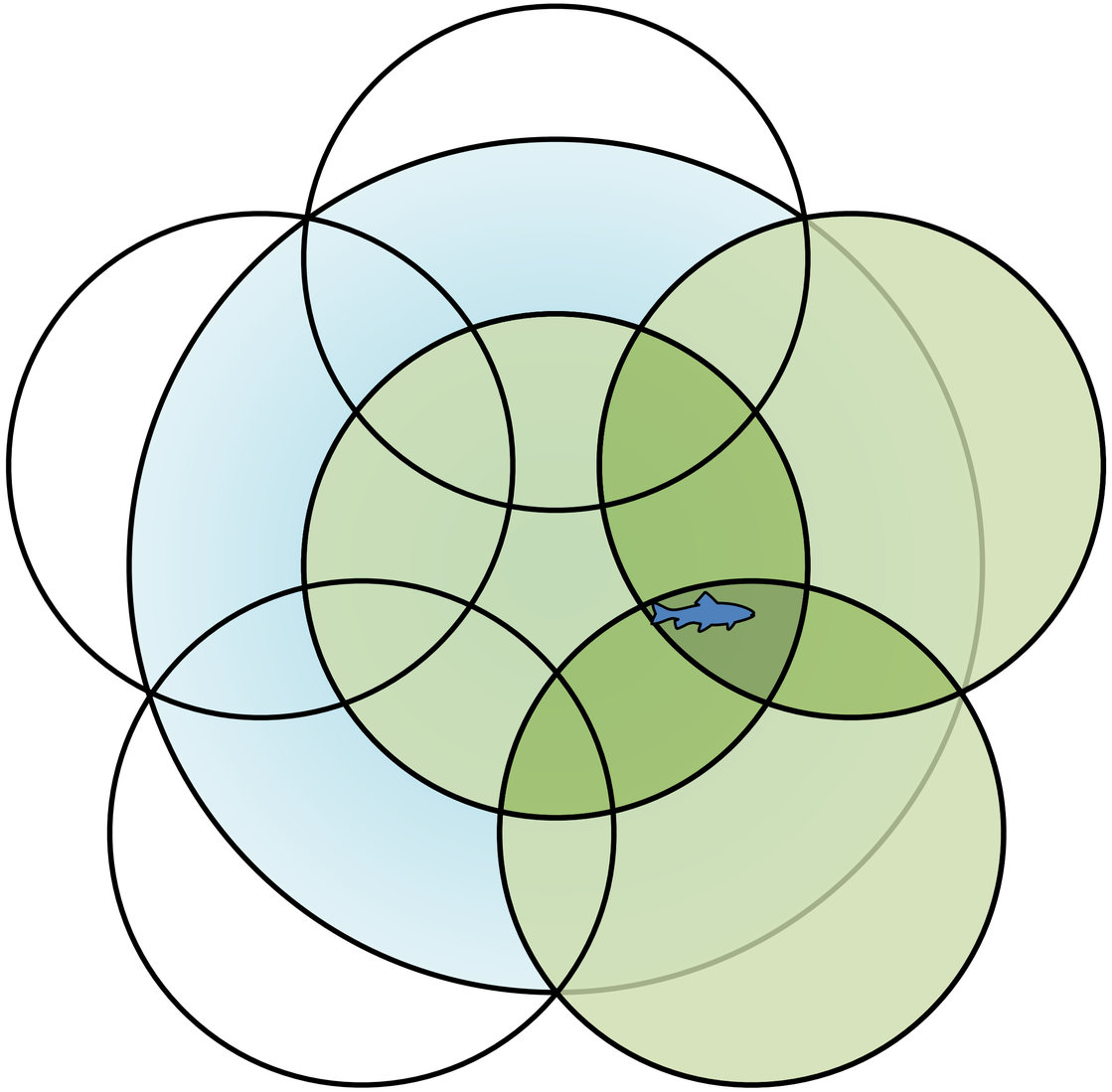
- **Linear function approximation**: how to process the input data to make a collection of features exploiting spacial locality
- Goals:
  - describe coarse coding
  - and describe how it relates to state aggregation.
    - we allow overlapping shapes
    - multiple "active" features
- What is the gradient of $\hat{v}(S_t, \mathbf{w})$ if the value function is linear in $\mathbf{w}$

Video 2: Generalization Properties of Coarse Coding

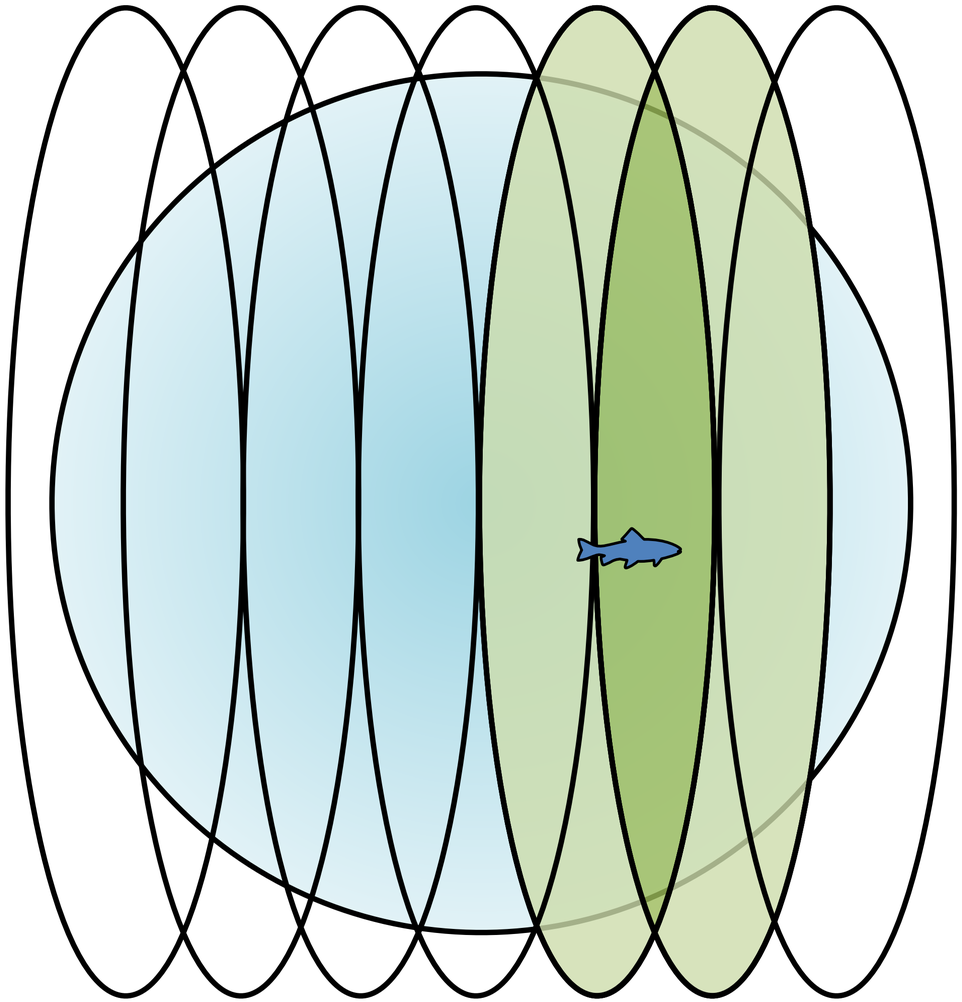
- **What do the shapes mean?**
- Goals:
  - Describe how coarse coding parameters affect generalization and discrimination
  - And understand how that affects learning speed and accuracy.
- In course coding what values can **x_t** take on?

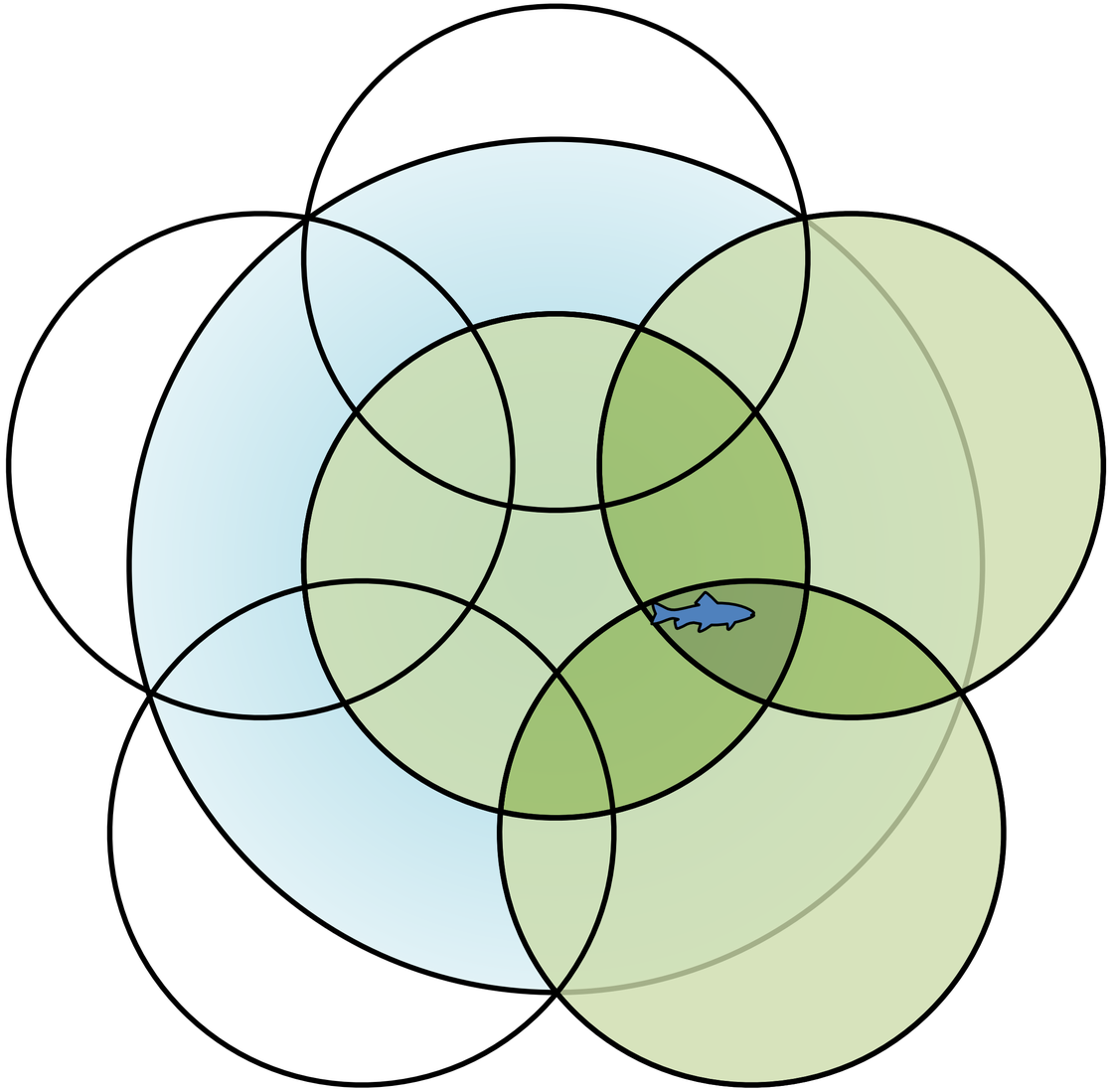
Broadness of generalization





Direction of generalization

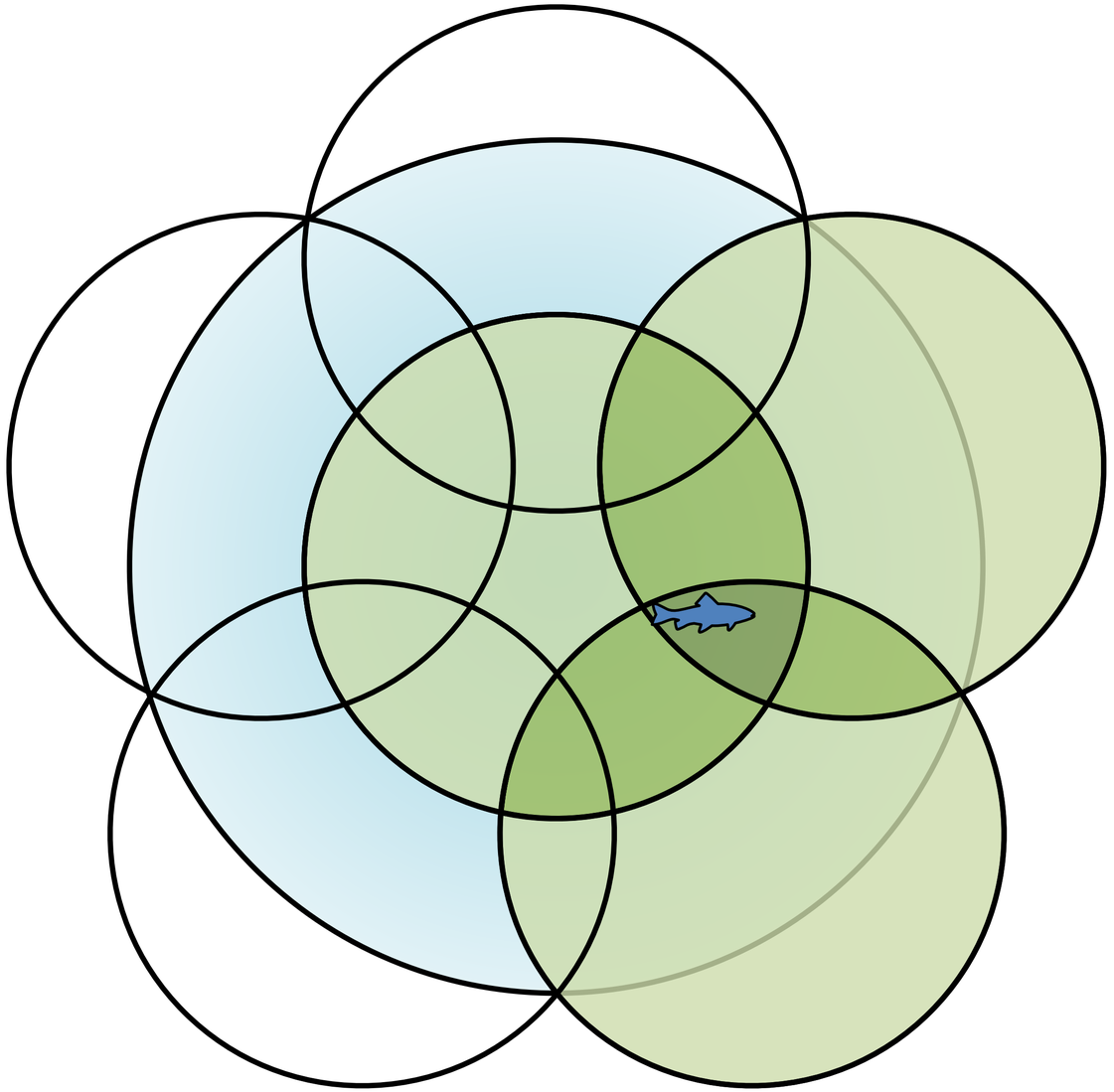




Number of features

- In the videos, we saw the example of how the size of a feature can affect generalization. Do the number and shape of a feature affect generalization in the same way?

- If we increase the number of circles, then how might this impact generalization and discrimination?



Video 3: Tile Coding

- A computationally efficient coarse coding scheme, that is incredibly useful for situations where the input is low dimensional (<=4). An interesting specific case of coarse coding!
- Goals:
  - explain how tile coding achieves both generalization and discrimination
  - and understand the benefits and limitations of tile coding
- In tile coding what values can **x_t** take on?
  - How many of them are non-zero?
- Which is more powerful (expressive), tile coding or state aggregation?
  - How can we make TC like state aggregation?

Video 4: Using Tile Coding for Prediction (video is not really about TD)

- Tile coding seems pretty great. Gradient Monte Carlo seems pretty good too. Let's put them together. A specific case of linear policy evaluation. We run an experiment on the thousand state chain, to see how tile coding compares to state aggregation

- Goals:

  - Explain how to use tile coding with linear prediction learning

  - And identify important properties of tile-coded representations.

- Why even use function approximation (tile coding) on the 1000 state chain?

Video 5: What is a Neural Network

- It is a non-linear function approximator, that learns the features. Note tile coding also provides non-linear value estimates in the state, but the value estimate is linear in the tile-coded features, making the gradient update for the weights simpler.
- Goals:
  - Understand feedforward neural networks
  - Define an activation function
  - And understand how a neural network is a parameterized function.
- Where are the features (**x**) in a NN?

Video 6: Non-linear Approximation with Neural Networks

- In tile coding we had a function to compute the features for an input state. A neural network learns what the features should be based on data. For a different problem it could learn different features. We call this representation learning!

- Goals:

  - Understand how neural networks do feature construction

  - And you will understand how neural networks are a nonlinear function of state.

- Representation learning is a big deal! Who designs the representation/features for TC?

  - Why is this approach limiting for AI?

Video 7: Deep Neural Networks

- It's a neural network with lots of layers. It is a popular network structure that has been very successful in practice. Results in powerful features, but can be more challenging to train.

- Goals:

  - Understand how deep neural networks are composed of many layers.

  - And Understand that depth can facilitate learning features, through composition and abstraction.

- Why have deep NNs become so popular? What is different from the 80’s to today to explain the success of NNs?

Video 8: How to compute the gradient

- Backprop is a procedure for computing the gradient of the NN and updating the weights (avoiding redundant computation). Its Stochastic Gradient Descent for a Neural Network function approximator.

- Goals:

  - derive the gradient of a neural network

  - and Implement gradient descent on a neural network

- What are some useful software libraries that can compute gradients for you automatically and help train NNs?

Video 9: Optimization Strategies for NNs

- Doing vanilla Stochastic Gradient Descent on a Deep NN does not work very well. In fact, doing SGD on a one layer NN doesn't work very well either.
- Part of all the recent progress is due to massive increases in data and compute
- But there have also been important algorithmic developments
- Goals:
  - Understand the importance of initialization for neural networks.
  - And describe optimization techniques for training neural networks (momentum and vector of step-sizes: adapting the learning rate based on data!)
- We have talked about scalar step-sizes, step-sizes decaying over time; what about having a step-size for every weight in the NN and changing them with time?

The Bitter lesson

- http://www.incompleteideas.net/IncIdeas/BitterLesson.html

- “The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin….

- …In computer chess, the methods that defeated the world champion, Kasparov, in 1997, were based on **massive, deep search**. At the time, this was looked upon with dismay by the majority of computer-chess researchers who had **pursued methods that leveraged human understanding of the special structure of chess**….

- … **The bitter lesson** is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning….”

Big Agent, Massive World

- Our current agents are huge!
- But the world is so much larger than any agent we can build
- However, our current agents operate in the over-parameterize regime
  - The number of parameterizes in the function approximator exceeds the training data (or number of states)
  - We use massive clusters to train huge architectures! And it works well it seems
- Will the continue to be the case as we move toward more ambitious applications? Towards AI?

Clarifications

- Adjustable parameters are just the weights in the network
- How should we design the tile coding: number of tilings and size of individual tiles
- How do we decide which activation function to use (ie relu)
- The number of hidden layers (depth) and the widths of the layers have a large impact on the neural network's performance. What is the general advice? Can this be done automatically?

Hyperparams

- As our agents get more complex they have more and more hyperparameters; what should we do to tune them?

More on tile coding