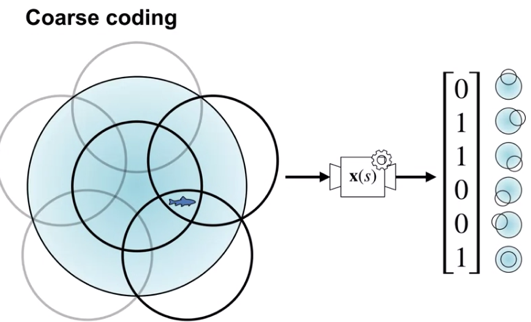
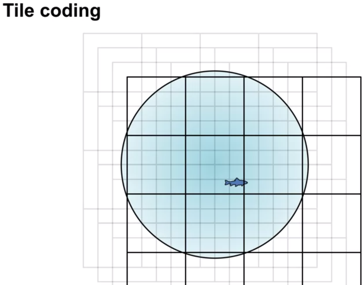
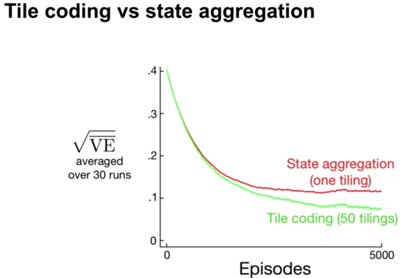
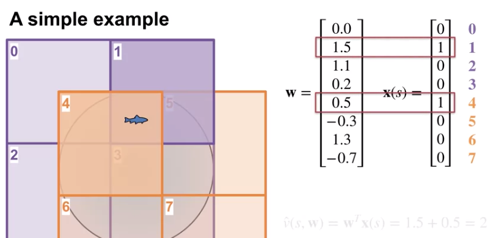
Week2 Notes

**Lesson 1: Feature Construction for Linear Methods**

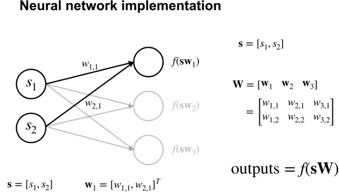
* Describe the difference between coarse coding and tabular representations
  + how the tabular case can be represented with a binary one hot encoding of the state and how course coding is a generalization of state aggregation.



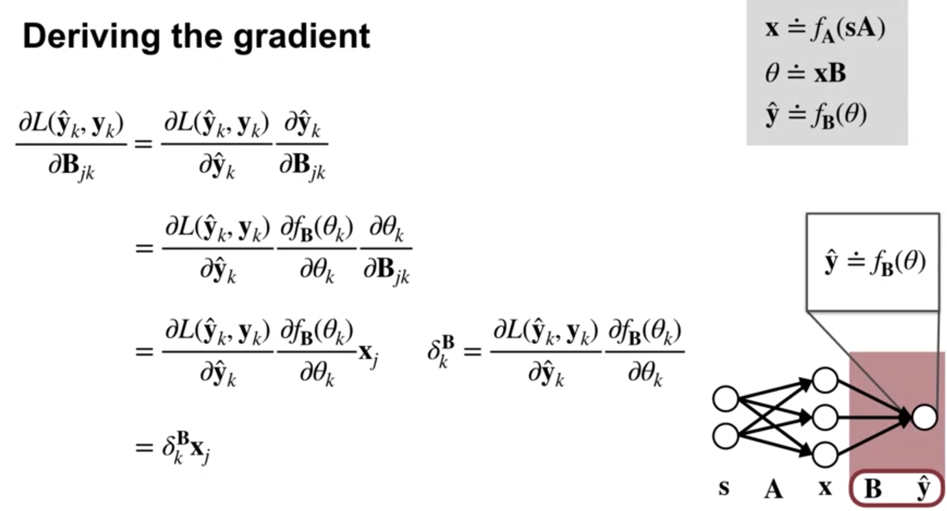
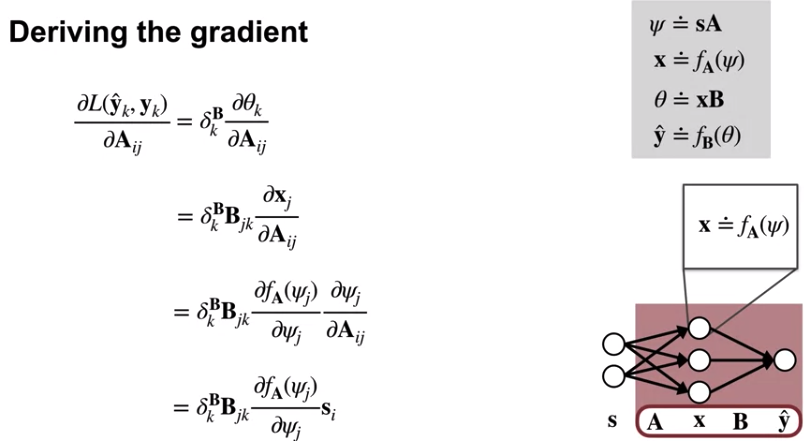
* Explain the trade-off when designing representations between discrimination and generalization
  + If union is large generalization is better. Shape and size of receptive fields impact generalization and speed of learning.
  + Ability to distinguish between values for 2 different states is called discrimination.
  + Overlap between the circles dictates level of discrimination. the size, number, and shape of the features all affect the discriminative ability of the representation
* Understand how different coarse coding schemes affect the functions that can be represented
* Explain how tile coding is a (computationally?) convenient case of coarse coding
  + 
  + Large tiles better generalization. Control generalization better with a grid of rectangles instead of squares.
  + Overlay several tiles better for discrimination
* Describe how designing the tilings affects the resultant representation
* Understand that tile coding is a computationally efficient implementation of coarse coding
  + Tile coding is also computationally efficient. Since grids are uniform, it's easy to compute which cell the current state is in. Due to its computational efficiency, tile coding can be used to quickly run preliminary experiments in low dimensional environments.

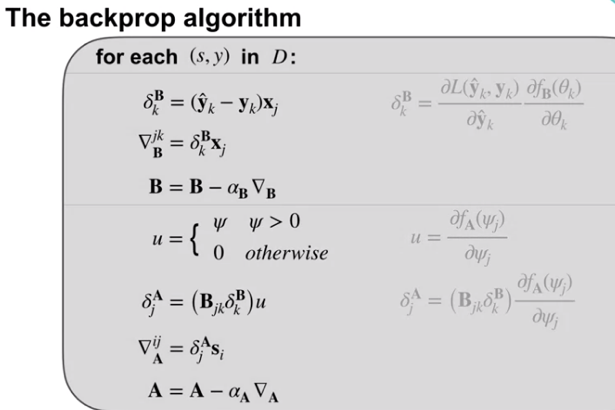
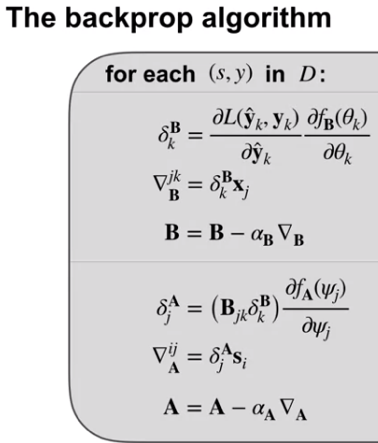


**Lesson 2: Neural Networks**

* Define a neural network
* 
* Define activation functions
* Define a feedforward architecture
* Understand how neural networks are doing feature construction
* Understand how neural networks are a non-linear function of state
* Understand how deep networks are a composition of layers
  + The depth of the network is defined by the number of hidden layers in the network. In theory, a neural network need not be deep. A neural network with a single hidden layer can approximate any continuous function given that is sufficiently wide. We call this the **universal approximation** property. However, practical experience and theory suggests that deep neural networks may make it easier to approximate complex functions. One reason for this is that the depth allows composition of features. Composition can produce more specialized features by combining modular components
  + Deep neural networks compose many layers of lower-level abstractions with each successive layer contributing to increasingly abstract representations. Introduce bottleneck layer.
* Understand the tradeoff between learning capacity and challenges presented by deeper networks

**Lesson 3: Training Neural Networks**

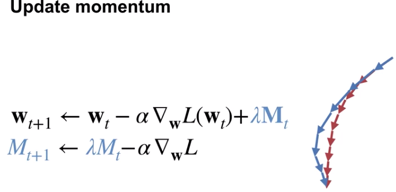
* Compute the gradient of a single hidden layer neural network
  + 
  + 
* Understand how to compute the gradient for arbitrarily deep networks



* Understand the importance of initialization for neural networks
* Describe strategies for initializing neural networks

One simple yet effective initialization strategy, is to randomly sample the initial weights from a normal distribution with small variance. This way, each neuron has a different output from other neurons within its layer. This provides a more diverse set of potential features. By keeping the variants small, we ensure that the output of each neuron is within the same range as its neighbors. One downside to this strategy is that, as we add more inputs to a neuron, the variance of the output grows. We can get around this issue by scaling the variance of the weights, by one over the square root of the number of inputs. After we have picked the starting point for our network, we start incrementally making small improvements to the weights using stochastic gradient descent steps.

* Describe optimization techniques for training neural networks
  + Momentum, Vector stepsize adaptation



separate step size for each weight in the network. So far, we have only talked about a global scalar step size. This is well-known to be problematic because this can result in updates that are too big for some weights and too small for other weights. Adapting the step sizes for each weight, based on statistics about the learning process in practice results in much better performance.

