**Artificial Intelligence in Games**

**Session 10**

1. Supervised learning: regression (dot means dot product)
   1. Consider an iid training dataset D = (x1, y1), . . . ,(xN, yN)
   2. Suppose xi in R^D and yi in R
   3. Regression: predicting target y given new observation x
   4. In our setting, xi will represent a state and yi will represent its (estimated) value
2. Orthogonality;
   1. A vector x in R^D is orthogonal to a vector w in R^D when:  
        
      w dot x = w1x1 + w2x2 + … + wDxD = sum of(w\_jx\_j) = 0  
      A hyperplane Sw is the set of vectors orthogonal to w: Sw = {x ∈ R D | w · x = 0}
3. Hyperplanes:
   1. A hyperplane S\_w is the set of vectors orthogonal to w:   
        
      S\_w = {x in R^D given w dot x = 0}
   2. For D = 2 or D = 3, a hyperplane is a line or plane that goes through the origin.
4. Linear regression: model
   1. Consider an iid dataset D = (x1, y1), . . . ,(xN, yN), where xi in R^D and yi in R
   2. Regression: predicting target y given new observation x
   3. Linear regression (naive model):
      1. Y = w dot x = sum of(w\_jx\_j)
   4. Interpretation: wj indicates how much each unit of xj contributes towards the target
   5. Linear regression (probabilistic model):
      1. Expectation of Y given x,w = w dot x
      2. p(y given x,w) = N(y given w dot x, sigma^2)
5. Linear regression: geometry for D = 1
   1. For any w, if y = wx, then wx − y = 0 and (w, −1) dot (x, y) = 0
   2. For any w, the pairs (x, y) for which y = wx constitute a hyperplane
      1. {(x, y) in R^2 given (w, −1) dot (x, y) = 0}  
           
         fig: xy graph with a vector w and data points plotted across in a linear way (y = x)
6. Linear regression: likelihood
   1. Assuming constant σ 2 , the conditional likelihood is given by:  
        
      p(D given w) = product sum of N(yi given w dot xi, sigma^2)
   2. The log-likelihood is given by:  
        
      log p(D given w) = -(N/2) log 2pi sigma^2 – (1/2 sigma^2) sum of (yi – w dot xi)^2
   3. Maximizing the likelihood wrt w corresponds to minimizing J given by  
        
      J(w) = 1/N sum of(yi − w dot xi) 2
   4. J can be minimized analytically (in non-degenerate cases)
7. Linear regression: extensions
   1. If w maximizes the likelihood, we may predict y = w dot x given x
      1. Alternative: maximum a posteriori estimate (requires a prior)
      2. Bayesian alternative: using a posterior predictive distribution
   2. Using a feature map phi: R^D 🡪 R^D’:
      1. p(y given x,w) = N(y given w dot phi(x), sigma^2)
   3. Bias-including feature map: phi(x) = (x,1)
      1. W dot phi(x) = w\_1:D dot x + w\_{D+1}
   4. Polynomial feature map (D = 1): phi(x) = (1, x1, …, x\_{D’-1})
      1. W dot phi(x) = sum of (w\_jx^{j-1})
8. Feedforward neural network:
   1. A feedforward neural network has a number of layers L
   2. This network may predict a target y = a^(L)\_1 given a new observation x = a^(1)
   3. Let N^(l) be the number of neurons in layer l
   4. The network has input neurons, hidden neurons, and output neurons
   5. Weighted input to neuron j in layer *l* > 1:
      1. z\_j^(1) = b\_j^(1) + sum of (w\_{j,k} a\_k)
   6. Activation of neuron j in layer 1 < l < L:
      1. a^(1) = sigma(z\_j)
   7. where sigma is a differentiable function, such as sigma(z) = 1/1+e^−z
   8. A feedforward neural network represents a parametric function of its weights and biases
   9. Let theta represent a joint assignment of weights and biases
   10. Maximizing the likelihood p(D given theta) corresponds to minimizing J given by
       1. J(theta) = 1/N sum of (y – a\_1)^2
   11. where a^(L)\_1 is the output of the network when a^(1) = x
   12. Minimization can be attempted by (stochastic) gradient descent or related techniques [Ruder, 2016]
   13. The gradient Nabla J(theta) can be computed using a technique called backpropagation
9. Gradient descent:
   1. Consider the task of minimizing f : R^D → R
   2. Gradient descent starts at an arbitrary estimate x0 in R^D and iteratively updates this estimate using x\_{t+1} = x\_t – n\_t Nabla f(x\_t), where n\_t is the learning rate at iteration t.
10. Generalization
    1. In large state spaces, some states may be seen very rarely
    2. In these cases, the state or action value estimates should generalize across states
    3. Generalization relies on function approximation
    4. The state value function V^pi: S → R can be approximated by a parametric function V : S × R^m → R
    5. The goal of policy evaluation becomes finding a parameter vector pi such that V^ pi (s) = V(s; theta), for every s in S
    6. Changing in changes the value estimates of several states
11. Example: linear value functions
    1. Suppose that any state s in S can be represented by a feature vector phi(s) in R^m, and that V(s; theta) is given by  
         
       V(s;theta) = theta dot phi(s) = sum of (theta\_i phi(s)\_i)
    2. Several model-free reinforcement learning algorithms work well in this case
12. Value regression
    1. For a given policy pi, consider a dataset D = {(si , V ^pi(si))}
    2. Value regression: predicting the value V^pi (s) of an unseen state s
    3. For a given parametric function V, the mean squared error J(theta) is given by   
         
       J(theta) = 1/N [V^pi (si) − V(si ; theta)]^2
    4. Let theta^\* denote a parameter vector such that J(theta^\*) = min\_theta J(theta)
    5. The function V(·; θ^\*) can be used to predict the value of unseen states
13. Stochastic gradient descent:
    1. Stochastic gradient descent is a procedure that converges to a local minimum of J (if the learning rate α is decayed appropriately)
    2. The procedure starts with an arbitrary estimate theta\_0
    3. For any t >= 0, a pair (st , V^pi (st)) is drawn at random from D, and the estimate theta\_{t+1} is obtained using:  
         
       theta\_{t+1} = theta\_t – 0.5\*alpha\*Nabla\_theta\*[V^pi(st) – V(st;theta\_t)]^2 where alpha is the learning rate
    4. By the chain rule,  
         
       theta\_t+1 = theta\_t + alpha[V^pi(st) – V(st;theta\_t)]\*Nabla\_theta\*V(st;theta\_t)
14. Value regression from estimates
    1. If V^pi (s) were available for all states s in S, there would be no need for function approximation
    2. Furthermore, in practice, a dataset will be given by D = {(si , vi)} where vi is an estimate of the value of si under policy pi
    3. Different estimates vi may be considered, such as the empirical return or one-step return observed after state si
    4. For any t >= 0, a pair (st , vt) is drawn at random from D, and the estimate θ\_{t+1} for stochastic gradient descent is obtained using:  
         
       theta\_t+1 = theta\_t + alpha[Vt – V(st;theta\_t)]\*Nabla\_theta\*V(st;theta\_t)
15. Gradient descent TD value estimation:
    1. Algorithm 1 Gradient descent temporal-difference value estimation algorithm
    2. Input: policy pi, number of episodes N, learning rate alpha, discount factor gamma
    3. Output: parameter vector theta
       1. Initialize pi arbitrarily
       2. for each i in {1, . . . , N} do
          1. s ← initial state for episode i
          2. while state s is not terminal do
             1. a ← pi(s)
             2. r ← observed reward for action a at state s
             3. s’ ← observed next state for action a at state s
             4. theta ← theta + alpha[r + gamma\*V(s’ ; theta) − V(s; theta)]\*Nabla\_theta\*V(s; θ)
             5. s ← s‘
          3. end while
       3. end for
16. Non-tabular model-free algorithms
    1. Q-learning control and Sarsa control can be adapted to use stochastic gradient descent to approximate action-value functions (rather than just value functions)
    2. The choice of an appropriate feature map phi : S → R^m is crucial to the success of any function approximation method
    3. In the case of linear approximation, it may be necessary to create features that are combinations of more natural features, since linear models are incapable of modeling relationships such as feature i being beneficial only in the absence of feature j
17. Q-learning control
    1. Algorithm 2 Q-learning control algorithm for linear function approximation
    2. Input: feature vector phi(s, a) for all state-action pairs (s, a), number of episodes N, learning rate alpha, exploration factor epsilon, discount factor gamma
    3. Output: parameter vector theta
    4. theta ← 0
    5. for each i in {1, . . . , N} do
       1. s ← initial state for episode i
       2. for each action a: do
          1. Q(a) ← P i thetai phi(s, a)i
       3. end for
       4. while state s is not terminal do
          1. if with probability 1 − epsilon: then
             1. a ← maximise argument of Q(a)
          2. else
             1. a ← random action
          3. end if
          4. r ← observed reward for action a at state s
          5. s’ ← observed next state for action a at state s
          6. delta ← r − Q(a)
          7. for each action a’ : do
             1. Q(a’) ← P i thetai phi(s’ , a’)i
          8. end for
          9. delta ← delta + gamma\*max of Q(a’) {Note: delta is the temporal difference}
          10. theta ← theta + alpha\* delta \*phi(s, a)
          11. s ← s’
       5. end while
    6. end for