Chapter 9 solutions **Hanna Gábor**

1. *Show that tabular methods such as presented in Part I of this book are a special case of linear function approximation. What would the feature vectors be?*

   The feature would be an indicator of the state: the feature vector's length would be equal to the number of states, and $x(s_i)$ would be a vector that contains 1 at the i^{th} coordinate and 0 at the others. The weight vector would also have the same length as the number of states. The i^th weight would be our value estimate for s_i. When we're doing the update, the gradient of $x(s_i)$ is 0 everywhere except for coordinate i, where it's 1, so we would only update the weight for the current state.

2. *Why does (9.17) define $(n+1)^k$ distinct features for dimension k?*

   For each state s_j, we choose one of $n+1$ options: $s_j^0, s_j^1 \ldots, s_j^n$. There are k states, we make a choice for each them: we can do that $((n+1)^k)$-many ways.

3. *What n and $c_{i,j}$ produce the feature vectors*

$$x(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2)^T?$$

$$\begin{aligned} n &= 2 \\ c_{1,1} &= c_{1,2} = 0 \\ c_{2,1} &= 1, c_{2,2} = 0 \\ c_{3,1} &= 0, c_{3,2} = 1 \\ c_{4,1} &= 1, c_{4,2} = 1 \\ c_{5,1} &= 2, c_{5,2} = 0 \\ c_{6,1} &= 0, c_{6,2} = 2 \\ c_{7,1} &= 1, c_{7,2} = 2 \\ c_{8,1} &= 2, c_{8,2} = 1 \\ c_{9,1} &= 2, c_{9,2} = 2 \end{aligned}$$

4. *Suppose we believe that one of two state dimensions is more likely to have an effect on the value function than is the other, that generalization should be primarily across this dimension rather than along it. What kind of tilings could be used to take advantage of this prior knowledge?*

   Let's say it's the first dimension that is more likely to have an effect on the value function. In this case, the tiles should be short in the first dimension and long in the second dimension. This way, if we learn from an example (s_1, s_2), we update many (s_1, x) states and less (x, s_2) states.

5. *Suppose you are using tile coding to transform a seven-dimensional continuous state space into binary feature vectors to estimate a state value function $\hat{v}(s, w) \approx v_\pi(s)$. You believe that the dimensions do not interact strongly, so you decide to use eight tilings of each dimension separately (stripe tilings), for $7 \cdot 8 = 56$ tilings. In addition, in case there are some pairwise interactions between the dimensions, you also take all $\binom{7}{2} = 21$ pairs of dimensions and tile each pair conjunctively with rectangular tiles. You make two tilings for each pair of dimensions, making a grand total of*

$21 \cdot 2 + 56 = 98$ tilings. Given these feature vectors, you suspect that you still have to average out some noise, so you decide that you want learning to be gradual, taking about 10 presentations with the same feature vector before learning nears its asymptote. What step-size parameter α should you use? Why?

Each state will be contained in 98 tiles, that means there will be 98 ones in the feature vector. Based on the rule of thumb before the exercise,

$$\alpha = \frac{1}{10\mathbb{E}(x^T x)} = \frac{1}{10 \cdot 98} = \frac{1}{980}$$

should be a good choice.

6. *If $\tau = 1$, prove that (9.19) together with (9.7) results in the error being reduced to zero in one update.*

   The goal is to prove that after the update $\mathbb{E}(x_t^T w_{t+1}) = \mathbb{E}(U_t)$.

$$\begin{aligned}
x_t^T w_{t+1} &= x_t^T (w_t + \alpha(U_t - \hat{v}(S_t, w_t))\nabla\hat{v}(S_t, w_t)) \\
&= x_t^T \Big(w_t + \frac{1}{\mathbb{E}(x^T x)}(U_t - x_t^T w_t)\Big) x_t \\
&= x_t^T w_t + U_t \frac{x_t^T x_t}{\mathbb{E}(x^T x)} - x_t^T w_t \frac{x_t^T x_t}{\mathbb{E}(x^T x)} \\
\mathbb{E}(x_t^T w_{t+1}) &= \mathbb{E}(x_t^T w_t) + \mathbb{E}\Big(U_t \frac{x_t^T x_t}{\mathbb{E}(x^T x)}\Big) - \mathbb{E}\Big(x_t^T w_t \frac{x_t^T x_t}{\mathbb{E}(x^T x)}\Big)
\end{aligned}$$

   If $x_t^T x_t$ is independent from U_t and $x_t^T w_t$, then $\mathbb{E}(x_t^T w_{t+1}) = \mathbb{E}(U_t)$ follows from this. Otherwise it doesn't. I'm not sure if the statement is true if these are not independent.

7. *One of the simplest artificial neural networks consists of a single semi-linear unit with a logistic nonlinearity. The need to handle approximate value functions of this form is common in games that end with either a win or a loss, in which case the value of a state can be interpreted as the probability of winning. Derive the learning algorithm for this case, from (9.7), such that no gradient notation appears.*

   The derivative of the sigmoid function σ is $\sigma'(x) = \sigma(x)(1 - \sigma(x)$.

$$\begin{aligned}
w_{t+1} &= w_t + \alpha\sigma(w_t^T x_t)\nabla\sigma(w_t^T x_t) \\
&= w_t + \alpha\sigma(w_t^T x_t)\sigma'(w_t^T x_t) x_t \\
&= w_t + \alpha\sigma(w_t^T x_t)^2(1 - \sigma(w_t^T x_t)) x_t
\end{aligned}$$

8. *Arguably, the squared error used to derive (9.7) is inappropriate for the case treated in the preceding exercise, and the right error measure is the cross-entropy loss (which you can find on Wikipedia). Repeat the derivation in Section 9.3, using the cross-entropy loss instead of the squared error in (9.4), all the way to an explicit form*

with no gradient or logarithm notation in it. Is your final form more complex, or simpler, than that you obtained in the preceding exercise?

It is a bit simpler in the end. For U_t and $\sigma(w_t^T x_t)$, the cross-entropy loss is

$$c = -(U_t \log(\sigma(w_t^T x_t)) + (1 - U_t)\log(1 - \sigma(w_t^T x_t))).$$

Using this and the derivative of σ:

$$\begin{aligned}
\nabla(c) &= -\frac{U_t}{\sigma(w_t^T x_t)}\sigma'(w_t^T x_t)x_t + \frac{(1-U_t)}{1-\sigma(w_t^T x_t)}\sigma'(w_t^T x_t)x_t \\
&= -\frac{U_t(1-\sigma(w_t^T x_t))}{\sigma(w_t^T x_t)(1-\sigma(w_t^T x_t))}\sigma'(w_t^T x_t)x_t + \frac{(1-U_t)\sigma(w_t^T x_t)}{\sigma(w_t^T x_t)(1-\sigma(w_t^T x_t))}\sigma'(w_t^T x_t)x_t \\
&= -U_t(1-\sigma(w_t^T x_t))x_t + (1-U_t)\sigma(w_t^T x_t)x_t \\
&= -U_t x_t + \sigma(w_t^T x_t)x_t \\
w_{t+1} &= w_t - \alpha\nabla(c) \\
&= w_t - \alpha(\sigma(w_t^T x_t) - U_t)x_t
\end{aligned}$$