CPSC 533V (2023W2 Term) Assignment 1

Yuwei Yin
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4
Student #: 36211928
yuweiyin@cs.ubc.ca

1 Reinforcement Learning Notation

Give expressions for each of the following.

(1.a) The return, i.e., the future cumulative rewards, G_t, when in state s at time t, for an MDP with deterministic dynamics and a deterministic policy, discount factor γ, rewards $r_t, r_{t+1}, \ldots$, and an infinite horizon. Use a summation.

$G_t =$

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} \tag{1.1}$$

where discount factor $0 \leq \gamma \leq 1$

(1.b) Rewrite your answer above in a recursive form, i.e., express G_t in terms of G_{t+1}.

$G_t =$

$$G_t = r_{t+1} + \gamma G_{t+1} \tag{1.2}$$

where discount factor $0 \leq \gamma \leq 1$

(1.c) The expected future cumulative rewards, $V(s)$, when in state s at time t, for an MDP with stochastic dynamics, discount factor γ, rewards r_t, and a fixed-horizon H.

$V(s) =$

$$V(s) = \mathbb{E}\left[\sum_{k=t}^{H} \gamma^{k-t} r_k \mid s_t = s\right] \tag{1.3}$$

where discount factor $0 \leq \gamma \leq 1$

(1.d) The expected return from state s, if the best-available action is taken in state s, given the current value function, $V(s)$, the stochastic dynamics, $P(s'|s,a)$, the rewards $r(s,a)$, the discount factor, γ, and the use of bootstrapping. Assume discrete states and discrete actions.

$V(s) =$

$$V(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')\right] \tag{1.4}$$

where discount factor $0 \leq \gamma \leq 1$

(1.e) The best action to take from state s, corresponding to the above question.

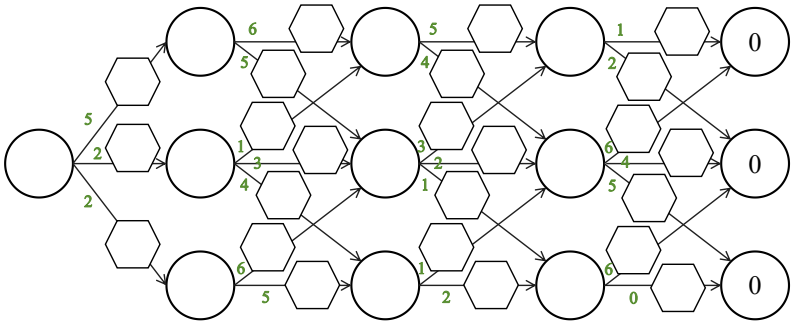
$V(s) =$

$$\pi(s) = \arg\max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')\right] \tag{1.5}$$

where discount factor $0 \leq \gamma \leq 1$

2 Deterministic MDP

Consider the following deterministic MDP. Assume no discounting. Assume zero rewards once the final states are reached.



(2.a) Compute the optimal state value function, $V(s)$, associated with each state. Write this in the circles that represent the states. Alternatively, list the values functions, in order from left-to-right, then top-to-bottom, i.e., a column-major order.

Please refer to Figure 1, where the green lines represent the optimal action of each state (circle).

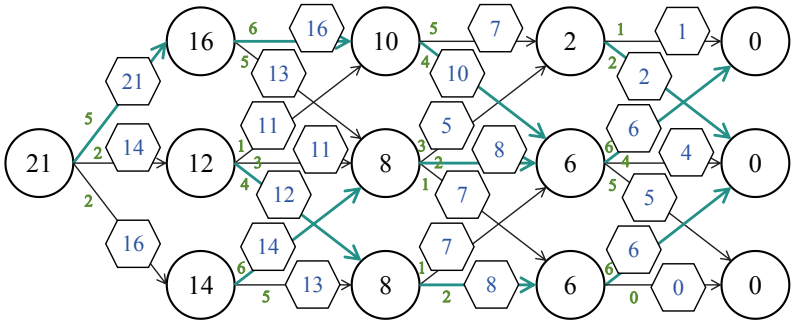


Figure 1: 533V Assignment 1 Q2 (a) solution.

(2.b) Annotate the optimal actions, using a heavier arrow. Alternatively, list the optimal actions for each state, implicitly numbered, e.g., $1, 2, 3$, and using a column-major order for the states.

Please refer to Figure 2, where the orange lines represent the optimal action path.

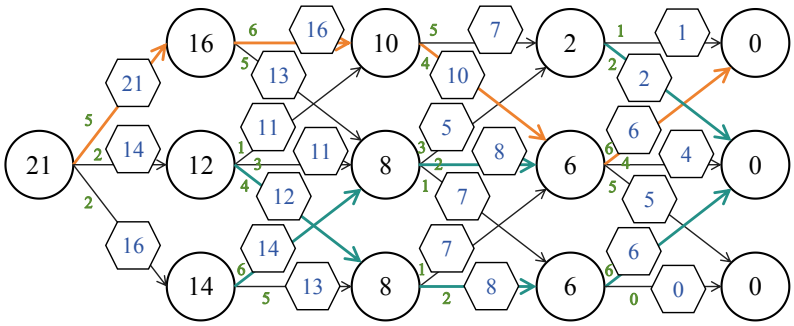


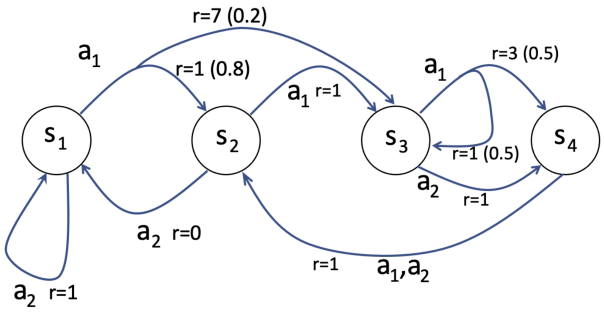
Figure 2: 533V Assignment 1 Q2 (b) (c) solution.

(2.c) Compute the optimal state-action function, $Q(s, a)$, associated with each state. Write this in the hexagons that represent all possible action transitions. Alternatively, list these values, in order from left-to-right, then top-to-bottom, i.e., a column-major order.

Please refer to Figure 2, where the hexagons represent values of the state-action function $Q(s, a)$.

3 Stochastic MDP

Consider the following stochastic MDP, and the given stochastic policy. Assume reward discounting, with $\gamma = 0.8$, and an infinite horizon. The bracketed number on the transitions indicate the probability of taking that given transition. All other transitions are taken with probability one.



state	$p(a_1)$	$p(a_2)$
s_1	0.8	0.2
s_2	0.5	0.5
s_3	1	0
s_4	0	1

stochastic policy $\pi(s)$

(3.a) Develop a transition probability matrix for the MDP, $P_{mn} : p(s_{t+1} = s_n | s_t = s_m)$ under the given policy $\pi(s)$. Element P_{mn} in column n thus represents the probability of transitioning from state m to state n. Also, develop a column vector R, where R_m is the expected immediate reward for the next transition when in state s_m, under the policy $\pi(s)$.

$$P_\pi = \begin{bmatrix} 0.2 & 0.64 & 0.16 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.1}$$

$$R_\pi = \begin{bmatrix} 0.2 \times 1 + 0.64 \times 1 + 0.16 \times 7 \\ 0.5 \times 0 + 0.5 \times 1 \\ 0.5 \times 1 + 0.5 \times 3 \\ 1 \times 1 \end{bmatrix} = \begin{bmatrix} 1.96 \\ 0.5 \\ 2 \\ 1 \end{bmatrix} \tag{3.2}$$

(3.b) The value function for the MDP under the policy π is equal to the immediate rewards, plus the discounted sum of future rewards, i.e.,

$$V(s) = \mathbb{E}_\pi \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right]$$

where $a = \pi(s)$. Give a linear algebra expression that is equivalent to this, using the P and R as determined previously. Then give a linear algebra expression that solves for $V(s)$. Lastly, solve for the value function using your favorite linear algebra package.

$$\begin{aligned} &\because V_\pi = R_\pi + \gamma P_\pi V_\pi \\ &\therefore V_\pi = (I - \gamma P_\pi)^{-1} R_\pi \\ &\therefore V_\pi = \begin{bmatrix} 7.37870117 \\ 6.42137348 \\ 7.42473252 \\ 6.13709878 \end{bmatrix} \end{aligned} \tag{3.3}$$

4 Reinforcement Learning

(4.a) Give three reasons why RL can be considered to be more difficult than supervised learning.

(1) sparse reward: In reinforcement learning, the AI agent receives sparse (~~continuous~~) and delayed (~~instant~~) rewards for the environment, which is more implicit for the model to learn. As for supervised

learning, the model is fed in labeled data, where the labels can be directly used to compute the learning loss.
(2) dynamic environment: The RL environment can be dynamic and changing, while the data distribution is assumed to be static in supervised learning.
(3) exploration-exploitation dilemma: RL agents need to balance between trying new actions to discover better strategies (exploration) and exploiting known strategies to maximize immediate rewards (exploitation). However, supervised learning usually does not need to deal with such trade-off.

(4.b) Are optimal policies unique? Why or why not?

No, optimal policies are not always unique. The uniqueness of optimal policies depends on various factors related to the environment, problem formulation, and the characteristics of the reinforcement learning task.
For example, in a deterministic environment where the outcome of actions is predictable, optimal policies are more likely to be unique. However, in stochastic environments with uncertain outcomes, multiple policies may lead to similar expected returns (thus may not be unique).

(4.c) Suppose that an MDP returns a reward of 1 at every time step. What is the expected return for the MDP for a discount factor of γ, and assuming an infinite horizon? Show your work. Give the expected returns for $\gamma = \{0, 0.5, 0.9, 0.99\}$. Give a definition for the **effective horizon** that can be associated with a discount factor. What discount factor should we use if we wish to have a policy that has an effective horizon of 5 time steps into the future?

As reward is always 1, the expected return with the infinite horizon is:

$$G = \sum_{t=0}^{\infty} \gamma^t = \lim_{t \to \infty} \frac{1 - \gamma^t}{1 - \gamma} = \frac{1}{1 - \gamma} \tag{4.1}$$

where discount factor $0 \leq \gamma \leq 1$

Therefore, for $\gamma = 0$, $G = \frac{1}{1-0} = 1$
For $\gamma = 0.5$, $G = \frac{1}{1-0.5} = 2$
For $\gamma = 0.9$, $G = \frac{1}{1-0.9} = 10$
For $\gamma = 0.99$, $G = \frac{1}{1-0.99} = 100$
And if horizon $H = 5 = \frac{1}{1-\gamma}$, we have $\gamma = 0.8$

(4.d) Given a discrete-state, discrete action MDP with $|S|$ states and $|A|$ actions, give an expression for the number of possible distinct policies. How many deterministic policies are there for the *deterministic MDP* given above? How many deterministic policies are there for the *stochastic MDP* given above?

For each discrete state, we choose a single action from the $|A|$ possible actions, thus the number of deterministic policies is $|A|^{|S|}$ for either deterministic or stochastic MDP.

5 Math Review: Evaluate the following Expressions

$$v = \begin{bmatrix} \pi \\ e \\ 1 \end{bmatrix}, w = \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix}, A = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}, \alpha = \frac{\pi}{4}$$

(5.a) $v^T w + \|v\|$ (2-norm)

$$v^T w + \|v\| = (2\pi + 3e + -1) + \sqrt{\pi^2 + e^2 + 1^2} \tag{5.1}$$

(5.b) $v^T A v$

$$
\begin{aligned}
v^T Av &= \begin{bmatrix} \pi\cos\alpha - \sin\alpha \\ e \\ \pi\sin\alpha + \cos\alpha \end{bmatrix}^T \begin{bmatrix} \pi \\ e \\ 1 \end{bmatrix} \\
&= (\pi^2\cos\alpha - \pi\sin\alpha) + e^2 + (\pi\sin\alpha + \cos\alpha) \\
&= (\pi^2+1)\cos\alpha + e^2 = (\pi^2+1)\cos\frac{\pi}{4} + e^2 \qquad (5.2) \\
&= \frac{\sqrt{2}}{2}(\pi^2+1) + e^2 \qquad (5.3)
\end{aligned}
$$

(5.c) $A^T A$

$$
\begin{aligned}
A^T A &= \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \\
&= \begin{bmatrix} \cos^2\alpha + \sin^2\alpha & 0 & \cos\alpha\cdot\sin\alpha - \sin\alpha\cdot\cos\alpha \\ 0 & 1 & 0 \\ \sin\alpha\cdot\cos\alpha - \cos\alpha\cdot\sin\alpha & 0 & \sin^2\alpha + \cos^2\alpha \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5.4)
\end{aligned}
$$

(5.d) $\min_{x\in[0,1]} x(1-x)$

$$
\begin{aligned}
&\because \arg\min_{x\in[0,1]} x(1-x) \in \{0,1\} \\
&\therefore \min_{x\in[0,1]} x(1-x) = 0 \qquad (5.5)
\end{aligned}
$$

(5.e) $\max_{x\in[0,1]} x(1-x)$

For $f(x) = x(1-x)$, $f'(x) = \frac{d}{dx}x(1-x) = 1-2x$. Let $f'(x) = 0$, we have $x = 0.5$

$$
\begin{aligned}
&\because \arg\max_{x\in[0,1]} x(1-x) = 0.5 \\
&\therefore \max_{x\in[0,1]} x(1-x) = 0.5^2 = 0.25 \qquad (5.6)
\end{aligned}
$$

(5.f) $\arg\max_{x\in[0,1]} x(1-x)$

$$
\arg\max_{x\in[0,1]} x(1-x) = 0.5 \qquad (5.7)
$$

6 Normal distribution derivatives

The stochastic policies we will use in reinforcement learning for continuous action problems will often use a normally-distributed action, with the policy output consisting of the mean, μ, and variance, σ^2, i.e., $a \sim \mathcal{N}(\mu, \sigma^2)$, where the normal distribution is given by:

$$
P(a) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{a-\mu}{\sigma})^2}
$$

We will see that to use back-propagation, we will need to know the derivative of the normal distribution probability with respect to μ and σ^2.

Show how to compute the derivatives of the normal distribution with respect to the mean and variance.

(1) Mean μ:

$$\begin{aligned}\frac{\partial}{\partial \mu} f(x|\mu,\sigma^2) &= \frac{\partial}{\partial \sigma^2}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\right) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\cdot\left(\frac{(x-\mu)}{\sigma^2}\right) \end{aligned} \tag{6.1}$$

(2) Variance σ^2:

$$\begin{aligned}\frac{\partial}{\partial \sigma^2} f(x|\mu,\sigma^2) &= -\frac{1}{2\sqrt{2\pi}(\sigma^2)^{3/2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \\ &\quad + \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\cdot\frac{(x-\mu)^2}{2(\sigma^2)^2} \end{aligned} \tag{6.2}$$

7 Understanding auto-differentiation

Deep reinforcement learning methods will make use of neural-networks to model the policy, $\pi(s)$, the state value function, $V(s)$, and the state-action value function, $Q(s,a)$. This question is about computation graphs and automatic differentiation. Training a neural network is about finding a set of parameters (weights) that optimizes a particular objective function. Gradient descent is key to this and thus automatic differentiation algorithms play a key role in being able to compute the gradients of loss functions with respect to neural network weights.
For the following vector function, answer the questions below.

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \mathbf{f}(x_1, x_2) = \begin{bmatrix} ln(x_1) + x_1 \sin(3x_2^2) \\ e^{x_1 x_2} + x_1 \end{bmatrix}$$

In the space below:
(7.a) Draw the computation graph.

Please refer to Figure 3, where the computation workflow is from left to right. x_1 and x_2 are the input nodes, y_1 and y_2 are the output nodes, and h_i is the i-th hidden-state node.

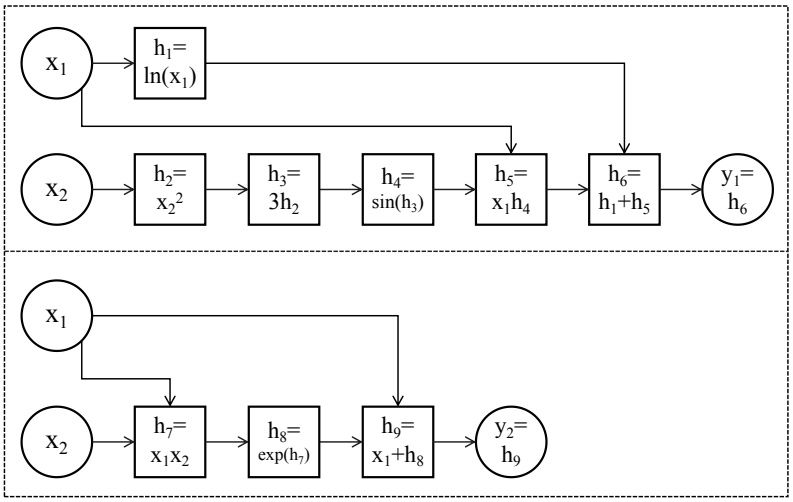


Figure 3: 533V Assignment 1 Q7 (a) solution.

(7.b) Annotate derivatives on each path of graph.

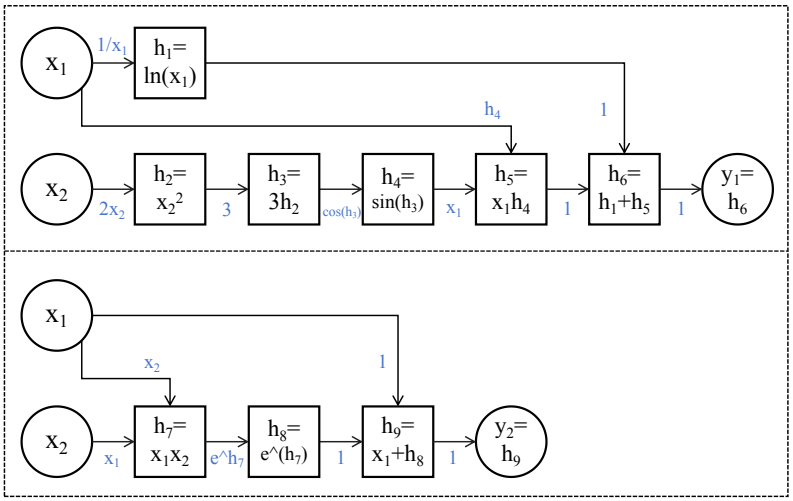


Figure 4: 533V Assignment 1 Q7 (b) solution.

Please refer to Figure 4, where the derivatives are annotated in blue color. For example, $h_5 = x_1 \cdot h_4$, so $\frac{\partial}{\partial x_1} h_5 = h_4$ and $\frac{\partial}{\partial h_4} h_5 = x_1$.

(7.c) Compute the value of $\mathbf{f}(x_1, x_2)$ at $x_1 = 1$ and $x_2 = 2$.

Forward pass with initialization of $x_1 = 1$ and $x_2 = 2$:

$$\begin{aligned} \mathbf{f}(x_1, x_2) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} &= \begin{bmatrix} ln(x_1) + x_1 \sin(3x_2^2) \\ e^{x_1 x_2} + x_1 \end{bmatrix} \\ &= \begin{bmatrix} ln(1) + 1 \times \sin(3 \times 2^2) \\ e^{1 \times 2} + 1 \end{bmatrix} \\ &= \begin{bmatrix} \sin(12) \\ e^2 + 1 \end{bmatrix} \end{aligned} \tag{7.1}$$

(7.d) Compute the Jacobian, $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$, at $x_1 = 1$ and $x_2 = 2$ using finite differences.

Apply the chain rule:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} \tag{7.2}$$

Here, we compute the four derivatives separately as follows:

$$\begin{aligned} \frac{\partial y_1}{\partial x_1} = \frac{\partial h_6}{\partial x_1} \times 1 &= \frac{\partial h_1}{\partial x_1} \times 1 + \frac{\partial h_5}{\partial x_1} \times 1 \\ &= \frac{1}{x_1} + \left(\frac{\partial x_1}{\partial x_1} \times h_4 + \frac{\partial h_4}{\partial x_1} \times x_1 \right) \\ &= \frac{1}{x_1} + \left(1 \times \sin(3x_2^2) + 0 \times x_1 \right) \\ &= \frac{1}{x_1} + \sin(3x_2^2) = 1 + \sin(12) \end{aligned} \tag{7.3}$$

$$\begin{aligned}\frac{\partial y_1}{\partial x_2} &= \frac{\partial h_6}{\partial x_2} \times 1 = \frac{\partial h_1}{\partial x_2} \times 1 + \frac{\partial h_5}{\partial x_2} \times 1 \\ &= 0 + \left(x_1 \times \frac{\partial h_4}{\partial x_2} \right) \\ &= x_1 \times \frac{\partial h_4}{\partial h_3} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_2}{\partial x_2} \\ &= x_1 \times \cos(h_3) \times 3 \times 2x_2 \\ &= 6x_1 x_2 \cos(3x_2^2) = 12\cos(12) \end{aligned} \tag{7.4}$$

$$\begin{aligned}\frac{\partial y_2}{\partial x_1} &= \frac{\partial h_9}{\partial x_1} \times 1 = \frac{\partial x_1}{\partial x_1} \times 1 + \frac{\partial h_8}{\partial x_1} \times 1 \\ &= 1 + \frac{\partial h_7}{\partial x_1} \times e^{h_7} \\ &= 1 + x_2 \cdot e^{x_1 x_2} = 1 + 2e^2 \end{aligned} \tag{7.5}$$

$$\begin{aligned}\frac{\partial y_2}{\partial x_2} &= \frac{\partial h_9}{\partial x_2} \times 1 = \frac{\partial x_1}{\partial x_2} \times 1 + \frac{\partial h_8}{\partial x_2} \times 1 \\ &= 0 + \frac{\partial h_7}{\partial x_2} \times e^{h_7} \\ &= x_1 \cdot e^{x_1 x_2} = e^2 \end{aligned} \tag{7.6}$$

Therefore, at $x_1 = 1$ and $x_2 = 2$ the Jacobian $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_1} \end{bmatrix} = \begin{bmatrix} 1 + \sin(12) & 12\cos(12) \\ 1 + 2e^2 & e^2 \end{bmatrix} \tag{7.7}$$

(7.e) Compute the Jacobian in closed form and evaluate it at the same point, i.e., $x_1 = 1$ and $x_2 = 2$.

At $x_1 = 1$ and $x_2 = 2$ the Jacobian matrix is:

$$\begin{aligned}\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_1} \end{bmatrix} &= \begin{bmatrix} \frac{1}{x_1} + \sin(3x_2^2) & 6x_1 x_2 \cos(3x_2^2) \\ 1 + x_2 \cdot e^{x_1 x_2} & x_1 \cdot e^{x_1 x_2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1} + \sin(3 \times 2^2) & 6 \times 1 \times 2 \times \cos(3 \times 2^2) \\ 1 + 2 \times e^{1 \times 2} & 1 \times e^{1 \times 2} \end{bmatrix} \\ &= \begin{bmatrix} 1 + \sin(12) & 12\cos(12) \\ 1 + 2e^2 & e^2 \end{bmatrix} \end{aligned} \tag{7.8}$$

8 Gradient descent

For $f(x) = x^2$, starting from $x = 2$, show how you can find the minimum of $f(x)$ using gradient descent step-by-step. Show the first two iterations using a gradient step size of 0.5.

Iteration 1 (time step t):
Initialization: Given the starting point $x_t = 2$ and learning rate $\alpha = 0.5$.
Compute the Gradient: Compute the gradient of $f(x)$ at $x_t = 2$: $f'(2) = 2 \times 2 = 4$
Update the Parameters: Update x using the gradient and the learning rate α:

$$\begin{aligned} x_{t+1} &= x_t - \alpha \times f'(x_t) \\ &= 2 - 0.5 = 0 \end{aligned} \tag{8.1}$$

Iteration 2 (time step $t+1$):
Initialization: Given the starting point $x_{t+1} = 0$ and learning rate $\alpha = 0.5$.
Compute the Gradient: Compute the gradient of $f(x)$ at $x_{t+1} = 0$: $f'(0) = 2 \times 0 = 0$
Update the Parameters: Update x using the gradient and the learning rate α:

$$\begin{aligned} x_{t+2} &= x_{t+1} - \alpha \times f'(x_{t+1}) \\ &= 0 - 0.5 = 0 \end{aligned} \tag{8.2}$$

In the second iteration, the parameter x remains the same because the gradient at $x = 0$ is 0, indicating that we have reached a minimum point of $f(x) = x^2$.

9 A hands-on exercise on a neural network's forward and backward pass.

Numerous good examples of forward and backward propagation for neural networks can be found online, e.g.,:
`https://theneuralblog.com/forward-pass-backpropagation-example/` A visual website to play with neural network architectures and weights is available here:
`https://playground.tensorflow.org/`

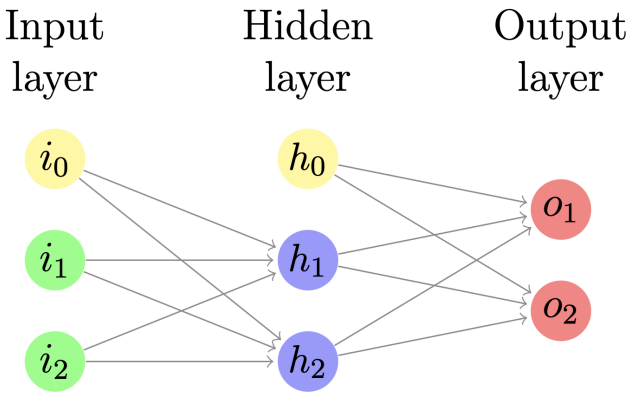


Figure 5: 533V Assignment 1 Q9. Neural network representation. i_0 and h_0 are the bias units.

Let us define a neural network with two inputs, two outputs, one hidden layer with two neurons, as depicted in Figure 1. Both layers have a bias of 1 (i_0 and h_0) and let's assume for simplicity that their weights are fixed. Use a ReLU activation on the hidden. The output layer is a linearly weighted combination of the hidden layer outputs. The loss function is L2. For updating the weights, we use stochastic gradient descent with a learning rate of 0.5.
The weights are initialized as follows:

$$\begin{aligned} w_{i_1-h_1} &= w_1 = 0.5 \\ w_{i_2-h_1} &= w_2 = 0.55 \\ w_{i_1-h_2} &= w_3 = 0.6 \\ w_{i_2-h_2} &= w_4 = 0.65 \\ w_{h_1-o_1} &= w_5 = 0.7 \\ w_{h_2-o_1} &= w_6 = 0.75 \\ w_{h_1-o_2} &= w_7 = 0.8 \\ w_{h_2-o_2} &= w_8 = 0.85 \\ w_{i_0-h_1} = w_{i_0-h_2} &= b_1 = 0.35 \\ w_{h_0-o_1} = w_{h_0-o_2} &= b_2 = 0.60 \end{aligned}$$

Use the following sample point:

$$\begin{aligned} X &= [0.3, 0.5] \\ Y &= [0.01, 0.99] \end{aligned}$$

For each of the following parts, show your work for at least one of the values requested.
For the others, the result is sufficient.

(9.a) Forward pass: Compute the values of o_1 and o_2 using X as input;

$$h_1 = 1 * 0.35 + 0.3 * 0.5 + 0.5 * 0.55 = 0.775 \tag{9.1}$$
$$h_2 = 1 * 0.35 + 0.3 * 0.6 + 0.5 * 0.65 = 0.855 \tag{9.2}$$
$$o_1 = 1 * 0.6 + 0.775 * 0.7 + 0.855 * 0.75 = 1.78375 \tag{9.3}$$
$$o_2 = 1 * 0.6 + 0.775 * 0.8 + 0.855 * 0.85 = 1.94675 \tag{9.4}$$

(9.b) Loss computation: Compute the total error E_{total} when the target value is Y;

The total error is the mean squared error loss:

$$\begin{aligned} E_{total} &= \frac{1}{2}\sum(y - \hat{y})^2 \\ &= \frac{1}{2}\left((0.01 - 1.78375)^2 + (0.99 - 1.94675)^2\right) \\ &= 2.0307798125 \end{aligned} \tag{9.5}$$

(9.c) Backward pass (output layer): Compute the partial errors with respect to the output layer $\frac{\partial E_{\text{total}}}{\partial w_5}$, $\frac{\partial E_{\text{total}}}{\partial w_6}$, $\frac{\partial E_{\text{total}}}{\partial w_7}$, $\frac{\partial E_{\text{total}}}{\partial w_8}$;

In the backward pass, we compute the gradients of the total error using the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_i} = \frac{\partial E_{\text{total}}}{\partial o_j} \cdot \frac{\partial o_j}{\partial w_i} \tag{9.6}$$

where o_j and w_i are the output and weight, respectively, of the j-th neuron in the i-th layer.

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial o_1}\frac{\partial o_1}{\partial w_5} = (o_1 - Y_1) \cdot h_1 = (1.78375 - 0.01) \times 0.775 = 1.37465625 \tag{9.7}$$
$$\frac{\partial E_{\text{total}}}{\partial w_6} = \frac{\partial E_{\text{total}}}{\partial o_1}\frac{\partial o_1}{\partial w_6} = (o_1 - Y_1) \cdot h_2 = (1.78375 - 0.01) \times 0.855 = 1.51655625 \tag{9.8}$$
$$\frac{\partial E_{\text{total}}}{\partial w_7} = \frac{\partial E_{\text{total}}}{\partial o_2}\frac{\partial o_2}{\partial w_7} = (o_2 - Y_2) \cdot h_1 = (1.94675 - 0.99) \times 0.775 = 0.74148125 \tag{9.9}$$
$$\frac{\partial E_{\text{total}}}{\partial w_8} = \frac{\partial E_{\text{total}}}{\partial o_2}\frac{\partial o_2}{\partial w_8} = (o_2 - Y_2) \cdot h_2 = (1.94675 - 0.99) \times 0.855 = 0.81802125 \tag{9.10}$$

(9.d) Optimization step (output layer): Using the values computed in the previous point, update the weights w_5, w_6, w_7, w_8;

Considering the learning rate $\alpha = 0.5$, we have the weight updating equation:

$$w' = w - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w} \tag{9.11}$$

$$w_5' = w_5 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_5} = 0.012671875 \tag{9.12}$$
$$w_6' = w_6 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_6} = -0.008278125 \tag{9.13}$$
$$w_7' = w_7 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_7} = 0.429259375 \tag{9.14}$$
$$w_8' = w_8 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_8} = 0.440989375 \tag{9.15}$$

(9.e) Backward pass (hidden layer): Compute the partial errors with respect to the hidden layer $\frac{\partial E_{\text{total}}}{\partial w_1}, \frac{\partial E_{\text{total}}}{\partial w_2}, \frac{\partial E_{\text{total}}}{\partial w_3}, \frac{\partial E_{\text{total}}}{\partial w_4}$;

We compute the gradients using the chain rule (Equation 9.6).

$$\begin{aligned}\frac{\partial E_{\text{total}}}{\partial w_1} &= \frac{\partial E_1}{\partial w_1} + \frac{\partial E_2}{\partial w_1} = i_1 \cdot ((o_1 - h_1) \cdot w_5 + (o_2 - h_1) \cdot w_7) \\ &= 0.3 \times ((1.78375 - 0.01) \times 0.7 + (1.94675 - 0.99) \times 0.8) \\ &= 0.6021075 \end{aligned} \tag{9.16}$$

$$\begin{aligned}\frac{\partial E_{\text{total}}}{\partial w_2} &= \frac{\partial E_1}{\partial w_2} + \frac{\partial E_2}{\partial w_2} = i_2 \cdot ((o_1 - h_1) \cdot w_5 + (o_2 - h_1) \cdot w_7) \\ &= 0.5 \times ((1.78375 - 0.01) \times 0.7 + (1.94675 - 0.99) \times 0.8) \\ &= 1.0035125 \end{aligned} \tag{9.17}$$

$$\begin{aligned}\frac{\partial E_{\text{total}}}{\partial w_3} &= \frac{\partial E_1}{\partial w_3} + \frac{\partial E_2}{\partial w_3} = i_1 \cdot ((o_1 - h_2) \cdot w_6 + (o_2 - h_2) \cdot w_8) \\ &= 0.3 \times ((1.78375 - 0.01) \times 0.75 + (1.94675 - 0.99) \times 0.85) \\ &= 0.643065 \end{aligned} \tag{9.18}$$

$$\begin{aligned}\frac{\partial E_{\text{total}}}{\partial w_4} &= \frac{\partial E_1}{\partial w_4} + \frac{\partial E_2}{\partial w_4} = i_2 \cdot ((o_1 - h_2) \cdot w_6 + (o_2 - h_2) \cdot w_8) \\ &= 0.5 \times)(1.78375 - 0.01) \times 0.75 + (1.94675 - 0.99) \times 0.85) \\ &= 1.071775 \end{aligned} \tag{9.19}$$

(9.f) Optimization step (hidden layer): Using the values computed in the previous point, update the weights w_1, w_2, w_3, w_4.

We compute the new weights using Equation 9.11.

$$w_1' = w_1 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_1} = 0.19894625 \tag{9.20}$$

$$w_2' = w_2 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_2} = 0.04824375 \tag{9.21}$$

$$w_3' = w_3 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_3} = 0.2784675 \tag{9.22}$$

$$w_4' = w_4 - \alpha \cdot \frac{\partial E_{\text{total}}}{\partial w_4} = 0.1141125 \tag{9.23}$$

10 Conceptual questions

This section contains conceptual questions. They will not be graded.

(a) Why do we randomly initialize the weights of a network network? Relevantly, what happens if we initialize all weights to 0? What happens if we initialize all weights to the same non-zero value?

- **Initialize all weight to 0**: If so, all neurons in the network would have the same output during the forward pass. Consequently, all neurons would update their weights in the same way during back-propagation. These neurons are deemed as dead neurons because they remain inactive and fail to learn unique features.
- **Initialize all weight to the same non-zero value**: If so, all neurons would learn the same features and have the same gradients during training, making the neurons behave the same in the same layer, interchangeable (symmetry among neurons), and thus incompatible to learn diverse representations and generalize well to different inputs.
- **Random Initialization**: Random initialization helps in preventing the aforementioned scenarios. (1) Randomly initializing weights introduces diversity in the learning process. Each neuron starts with a different set of weights, which allows them to learn different features from the input data. This diversity helps the network explore a broader range of features during training. (2) Random initialization helps the optimization algorithm converge faster. If all weights start with the same value, the neurons would move in the same direction during back-propagation, leading to slow convergence. Randomly initialized weights enable faster convergence by allowing neurons to explore different directions in the weight space.

(b) What happens if we were to use a purely linear activation function?

The purpose of activation functions is to introduce non-linearity to the model. Without non-linear activation functions, the entire network would be equivalent to a single linear transformation.

(c) Why do we use stochastic gradient descent instead of gradient descent?

Stochastic Gradient Descent (SGD) is often preferred over traditional Gradient Descent for several reasons, such as better computational efficiency, exploration with randomness, and the regularization effect. It is also worth noting that SGD can escape from saddle points more easily than pure batch Gradient Descent.

(d) Why is ReLU more commonly used than a sigmoid?

ReLU activation function is more commonly used for several reasons, such as ReLU is more computational efficient because it is easier to calculate the gradient. In addition, ReLU avoids the vanishing gradient problem in Sigmoid because the derivatives of Sigmoid are between 0 to 0.25.

11 Reading

Read the paper [1]: "A critique of pure learning and what artificial neural networks can learn from animal brains" `https://www.nature.com/articles/s41467-019-11786-6.pdf`

(a) Give two implications (of the many possible implications) of the proposed point-of-view for reinforcement learning.

This paper compares the learning process between artificial neural networks (ANNs) and animals. It is interesting to have these analogies in the neuroscience and evolutionary perspective. Thus, I read the paper carefully.

The paper argues that the main reason that animals function so well so soon after birth is that they rely heavily on innate mechanisms. Innate mechanisms, rather than heretofore undiscovered unsupervised learning algorithms, provide the base for Nature's secret sauce. Most of the data that contribute an animal's fitness are encoded by evolution into the genome. Also, the genome doesn't encode representations or behaviors directly (like synaptic weights); it encodes wiring rules and connection motifs. As for the proposed point-of-view for reinforcement learning, there are some implications (RL is not the main focus in this paper):

- 1. The paper deems evolution as a kind of reinforcement algorithm, operating on the timescale of generations, where the reinforcement signal consists of the number of progeny an individual generates.
- 2. The reinforcement learning algorithms underlying recent successes such as AlphaGo Zero draw their inspiration from the study of animal learning.

(b) Give an argument that goes at least partly against the given point of view. Keep your answers short, i.e., one paragraph for each implication and each argument. Be original!

I largely agree with the ideas in the paper. If I have to give (partly) opposing arguments:

- 1. The generalization (or analogy) from the rewards of RL to natural selection and genome passing is a bit far-fetched.
- 2. From what I see, the inspiration and intuition of RL (especially recent RL) have less to do with the idea of animal learning.

References

[1] Anthony M Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):3770, 2019.