Lecture 8

10.11.2025

Today's plan and announcements

- Hour 1: Reinforcement learning through policy gradients
- Hour 2: recurrent neural networks

- Exercise hour this week
  - Problem set 4
  - You can also ask questions on Problem set 3 and the python homework

Dynamical system review

So far, we looked at 3 classes of dynamical systems

- Continuous-time deterministic: $\dot{s}(t) = f(s(t), a(t))$
  - You have seen this in your dynamics and control systems course

- Discrete-time deterministic: $s_{t+1} = f_d(s_t, a_t)$
  - Can be obtained by discretization (e.g. Euler discretization) of a continuous-time system
  - You will see this in digital control course

- Discrete-time stochastic: $s_{t+1} \sim P(\cdot \mid s_t, a_t)$
  - Above is also known as a Markov decision-process (MDP)
  - This is the framework used in reinforcement learning (our focus in this course)

Dynamical system control

- Focusing on the Markov decision process, our control goal is to find a policy: $\pi : S \to A$

discount factor

$$\max_{\pi} J(\pi) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t = \pi(s_t), s_{t+1} \sim P(\cdot \,|\, s_t, a_t)\right]$$

discounted

- The above is an infinite horizon expected reward to maximize. , $\gamma \in (0,1)$ chosen.

- Why expectation $\mathbb{E}$? The transition dynamics are stochastic (later, we will also introduce stochastic policies)

suppose $\exists M \geq 0$ s.t. $\forall s,a \quad |r(s,a)| \leq M$

$$\sum_{t=0}^{\infty} \gamma^t r(s,a) \leq \sum_{t=0}^{\infty} \gamma^t M = \frac{1}{1-\gamma} M$$

- The discount factor γ ensures the expected reward is finite if $r(s,a)$ is bounded
- We can consider also finite-horizon objectives and more general goals such as stability, safety, reachability (subject of more advanced courses).

Stochastic control objective and policy parameterization

- Dynamic programming theory characterizes the optimal policy. But we aim for a learning approach to the problem in this course, in the spirit of machine learning.

$$J(\pi) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t = \pi(s_t), s_{t+1} \sim P(\,\cdot\,|\, s_t, a_t)\right]$$

- Policy Optimization: parameterize a stochastic policy as $\pi_\theta(\,\cdot\,|\, s)$, where $\theta \in \mathbf{R}^d$. The expected return of the policy is

$$\max_\theta J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\,\cdot\,|\, s_t), s_{t+1} \sim P(\,\cdot\,|\, s_t, a_t)\right]$$

Policy gradient approach

- Objective to be optimized

$$\max_{\theta} J(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 \sim \rho, a_t \sim \pi_\theta(\cdot \,|\, s_t), s_{t+1} \sim P(\cdot \,|\, s_t, a_t)\right]$$

- Now, we have a finite dimensional optimization problem. Use gradient ascent to try to find the best policy parameter.

$K \leftarrow$ number of training iterations, θ initialized randomly, $\alpha \leftarrow$ step size
for $i = 1, 2, \ldots, K$ **do**
  Calculate the gradient $\nabla_\theta J(\pi_\theta)$
  $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

$\left(\theta_{i+1} = \theta_i + \alpha \nabla_\theta J(\pi_{\theta_i}) \right)$

- Next, how do we compute the gradients?

Policy gradient theorem

$$\nabla_\theta J(\pi_\theta) \approx \nabla_\theta J_H(\pi_\theta) = \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

finite horizon approximation of objective: $J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right]$

- Above, H is referred to as the horizon. The first approximation is due to replacing infinite with finite horizon trajectory
- We proved the following two facts last time.

$\tau_H = (s_0, a_0, s_1, a_1, \cdots, s_H, a_H, s_{H+1})$

Fact 1: probability of the trajectory: $p_\theta(\tau_H) = \rho(s_0) \prod_{t=0}^{H} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$. (from Markov property)

Fact 2: $\nabla_\theta p_\theta(\tau) = \nabla_\theta p_\theta(\tau) \frac{p_\theta}{p_\theta} = \nabla_\theta \log p_\theta(\tau) p_\theta(\tau)$.

We will now use them to derive the policy gradient

Expected reward of a trajectory

- Reward from trajectory $\tau_H = \left(s_0, a_0, s_1, a_1, \ldots, s_H, a_H, s_{H+1}\right)$

$$R\left(\tau_H\right) := \sum_{t=0}^{H} \gamma^t r(s_t, a_t)$$

; $p_\theta(\tau_H)$: probability of trajectory τ_H

- Expected reward:

$$J_H(\pi_\theta) := \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \,|\, s_0 = s, \pi\right] = \mathbb{E}_{\tau_H \sim p_\theta}[R(\tau_H)]$$

$p_\theta(\tau_H)$

- We need to compute $\nabla_\theta J_H(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau_H \sim p_\theta}[R(\tau_H)]$

$p_\theta(\tau_H)$

Deriving the policy gradient

- Need to compute $\nabla_\theta J_H(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau_H \sim p_\theta}[R(\tau_H)]$

$$= \nabla_\theta \sum_{\tau_H} R(\tau_H) P_\theta(\tau_H)$$

As differentiation is a linear operator.

$$= \sum_{\tau_H} R(\tau_H) \nabla_\theta P_\theta(\tau_H)$$

using fact 2.

$$= \sum_{\tau_H} R(\tau_H) \nabla_\theta \log(P_\theta(\tau_H)) P_\theta(\tau_H) \quad \text{using Fact 1}$$

$$= \sum_{\tau_H} R(\tau_H) \nabla_\theta \log\left(\rho(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)\right) P_\theta(\tau_H)$$

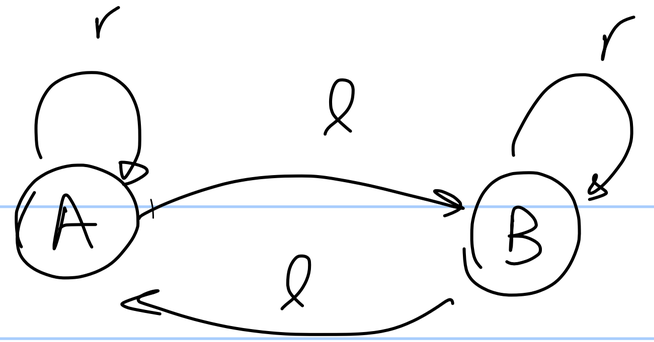
$$= \sum_{\tau_H} R(\tau_H) \sum_{t=0}^{H} \nabla_\theta \log(\rho(s_0)) + \nabla_\theta \log(\pi_\theta(a_t|s_t)) + \nabla_\theta \log P(s_{t+1}|s_t, a_t) P_\theta(\tau_H)$$

($\nabla_\theta \log(\rho(s_0)) = 0$, $\nabla_\theta \log P(s_{t+1}|s_t, a_t) = 0$)

$$= \sum_{\tau_H} R(\tau_H) \sum_{t=0}^{H} \nabla_\theta \log(\pi_\theta(a_t|s_t)) P_\theta(\tau_H)$$

$$= \mathbb{E}_{P_\theta(\tau_H)}\left[R(\tau_H) \sum_{t=0}^{H} \nabla_\theta \log(\pi_\theta(a_t|s_t))\right]$$

Policy gradient example



- Consider an MDP with 2 states $\{A, B\}$, and 2 actions $\{r, l\}$. In each state, taking the action r (remain) leads to remaining in the state and the action l (leave) leads to transitioning to the other state.
- The reward is defined as $r(A, r) = 1, r(A, l) = 0, r(B, r) = 0, r(B, l) = 2$, and discount factor is $\gamma = 0.9$.
- Let the probability of taking action r in state A and B be p_A and p_B, respectively.
- We consider a short episode with two time steps ($H = 1$). So, we have a trajectory, $(s_0, a_0, s_1, a_1, s_2)$, with $s_i \in \{A, B\}, a_i \in \{r, l\}$. Fill out the table below. Assume $s_0 = A$.

$(r(s_0, a_0), r(s_1, a_1))$

trajectory	probability	reward	Expected return
(A, r, A, r, A)	p_A^2	$(1, 1)$	$1 + (0.9)1 = 1.9$
(A, r, A, l, B)	$p_A(1 - p_A)$	$(1, 0)$	$1 + (0.9)0 = 1$
(A, l, B, r, B)	$(1 - p_A)\, p_B$	$(0, 0)$	$0 + (0.9)0 = 0$
(A, l, B, l, A)	$(1 - p_A)(1 - p_B)$	$(0, 2)$	$0 + (0.9)2 = 1.8$

$\gamma^0 r(s_0, a_0) + \gamma^1 r(s_1, a_1)$

Policy gradient example: direct parameterization gradients

- Formulate the expected reward over the two time steps. Your final solution should look like

$$J(p_A, p_B) = 1.8 - 0.8\,p_A - 1.8\,p_B + 0.9\,p_A^2 + 1.8\,p_A p_B\,.$$

$= p_A^2\,(1.9) + p_A(1-p_A)(1) + (1-p_A)\,p_B \cdot 0 + (1-p_A)(1-p_B)(1.8)$

- Verify the gradient of the expected reward over two time-steps is as follows.

$$\frac{\partial J}{\partial p_A} = -0.8 + 1.8\,p_A + 1.8\,p_B, \qquad \frac{\partial J}{\partial p_B} = -1.8 + 1.8\,p_A\,.$$

$J(p_A, p_B)$

- Is the above function convex? Hint: derive the Hessian $H(p_A, p_B) = \begin{bmatrix} \frac{\partial^2 J}{\partial p_A^2} & \frac{\partial^2 J}{\partial p_A \partial p_B} \\ \frac{\partial^2 J}{\partial p_B \partial p_A} & \frac{\partial^2 J}{\partial p_B^2} \end{bmatrix}$.

$= \begin{bmatrix} 1.8 & 1.8 \\ 1.8 & 0 \end{bmatrix}$

sum of eigenvalues

product of eigenvalues

$(\lambda - 1.8)\,\lambda - (1.8)^2 = 0$

$\lambda^2 - 1.8\,\lambda - (1.8)^2 = 0$

eigenvalues are root of this second order polynomial.

$\Rightarrow$ non-convex, since root are negative & positive

Policy gradient example: optimal parameters in direct parameterization

- Given the objective $J(p_A, p_B) = 1.8 - 0.8\,p_A - 1.8\,p_B + 0.9\,p_A^2 + 1.8\,p_A p_B$, and the gradients $\frac{\partial J}{\partial p_A} = -0.8 + 1.8\,p_A + 1.8\,p_B$, $\frac{\partial J}{\partial p_B} = -1.8 + 1.8\,p_A$, what would be the optimal parameters? Hint: a continuous function over a compact (closed and bounded interval) attains its optimum either at the point where the gradient vanishes or at the boundary points.

$p_A, p_B \in [0,1]$

$\frac{\partial J}{\partial p_B} = 0 \Rightarrow p_A = 1$, $\frac{\partial J}{\partial p_A} = 0 \Rightarrow p_B = \frac{-1}{1.8} \notin [0,1]$

so, the maximum in the interval $[0,1]_{p_A} \times [0,1]_{p_B}$ is not where the gradient vanishes.

$\Rightarrow$ let's check boundaries: $p_A = p_B = 1$, $p_A = 1, p_B = 0$

$p_A = p_B = 0$, $p_A = 0, p_B = 1$

$J(p_A, p_B)$'s maximum is at $p_A = 1$, p_B arbitrary.

Policy gradient example, softmax parameterization

Exercise.

- Assume that $p_A = \frac{e^{\theta_1}}{1 + e^{\theta_1}}$, and $p_B = \frac{e^{\theta_2}}{1 + e^{\theta_2}}$ (Note: softmax for 2 actions reduces to the sigmoid function - see our lecture on logistic regression). Compute the gradient of the expected return with respect to the parameters. Hint: First, verify that $\frac{\partial p_s}{\partial \theta_i} = p_s(1 - p_s),$ for $s \in \{A, B\}$. Then, use the chain rule and your previously computed gradients.

- Now, verify the policy gradient theorem would give you the same answer as above. Hint: First, verify that $\nabla_{\theta_i} \log \pi_{\theta_i}(r \,|\, s) = (1 - p_s), \quad \nabla_{\theta_i} \log \pi_{\theta_i}(l \,|\, s) = -p_s,$ for $s \in \{A, B\}$. Then, use this with the probability of the trajectories you computed from the table.

Estimating gradient from samples (learning)

- Why is policy gradient theorem useful?
  - If we don't know the model (RL setting), we can empirically estimate the gradients from trajectories

  - Sample N trajectories by interacting with the system or a simulator (Think of a robotic simulator)

$$\left(s_0^i, a_0^i, s_1^i, a_1^i, \ldots, s_H^i, a_H^i, s_{H+1}^i\right), \quad i = \{1, \ldots, N\}$$

  - Empirical estimate of $\nabla_\theta J(\pi_\theta)$

From policy gradient theorem

$$\nabla_\theta J(\pi_\theta) \approx \nabla_\theta J_H(\pi_\theta)$$

$$= \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{H} \gamma^t r(s_t^i, a_t^i)\right)\left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)$$

Policy gradient with gradient estimator

$K \leftarrow$ number of training iterations, θ initialized randomly, $\alpha \leftarrow$ step size
for $i = 1, 2, \ldots, K$ **do**
~~Calculate the gradient $\nabla_\theta J(\pi_\theta)$~~
$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

Sample trajectories $\{\tau_H^i = (s_0^i, a_0^i, \ldots, s_{H+1}^i)\}$ for $i = 1, \ldots, N$

estimate $\nabla_\theta J(\pi_\theta)$ using Monte Carlo.

- Convergence under certain conditions on the Markov decision process and/or policy parameterization
- In practice, requires very large number of samples for good gradient estimation and convergence -> you will see in your python homework
- Advise: If you have a model of the system, model-based methods are better -> optimization, (approximate) dynamic programming, model predictive control

Policy gradient example, starting with deterministic policies

Exercise .

- Assume that $p_A = 1, p_B = 1$. Using these probabilities, we sample the trajectories. What trajectory you get if you start in state A? What if you start in state B?

- What would the policy gradient from the sampled trajectories look like? Recall:

$$\nabla_\theta J_H(\pi_\theta) = \mathbb{E}_{\tau_H \sim p_\theta}\left[\left(\sum_{t=0}^{H} \gamma^t r(s_t, a_t)\right) \times \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\right]$$

- What is the problem with starting with deterministic policies?

Summary - reinforcement learning

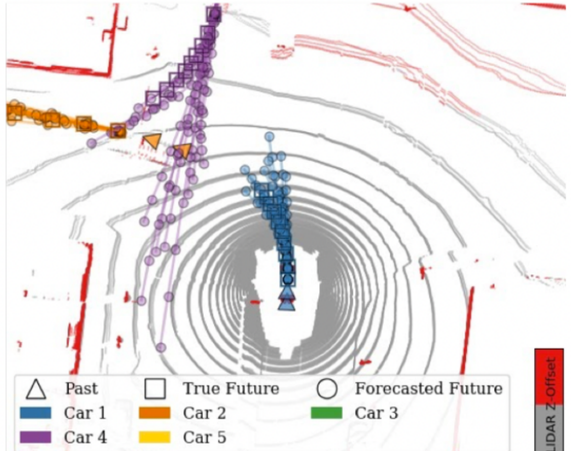
- Reinforcement learning is a learning-based approach to compute (often approximate) solutions to discrete-time optimal control problems
- One approach to reinforcement learning is policy gradient
  - Parameterizing the control policy
  - Compute gradient of the expected reward with respect to the parameters
  - Update the parameters in the direction of ascent to improve the reward

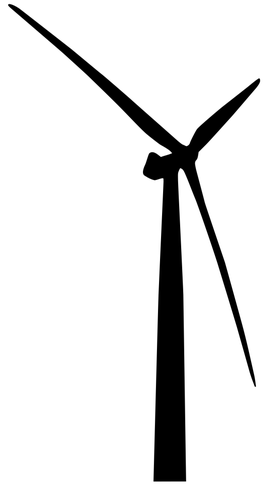
- Policy gradient theorem
  - characterizes gradient as an expectation -> gradient can be estimated from Monte Carlo samples of the system trajectory
  - Has shown some success but usually requires a very large number of samples even for small

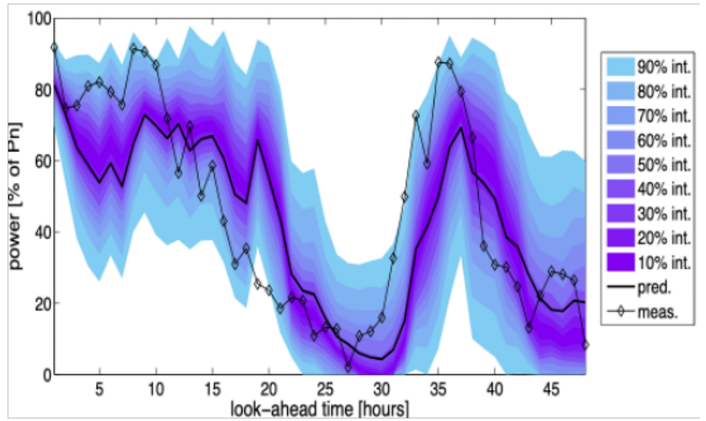
Recurrent neural networks

Motivation

- Consider the following applications
  - Time-series prediction: weather forecasting, price signals
  - Dynamical system trajectory/output prediction
  - Natural language processing (example: Large language models)
- Above, we have a time-series: a signal that is changing over time. Given the past signal values, we want to predict the future
  - Think of auto-compete on your phones







Recurrent neural network motivation

- For time-series prediction, we need to exploit the time structure to efficiently do the prediction

- Recurrent neural networks are simply dynamical systems - the type you have been looking at for the past few lectures.

- Unlike the lecture on control/RL, our goal is not to find the optimal input policy, rather to predict the output given the input.

Recurrent neural network examples

(RNN)

- Consider a dynamical system with state $s \in \mathbb{R}^{n_s}$, input $a \in \mathbb{R}^{n_a}$, and output $y \in \mathbb{R}^{n_o}$:

$$s_{t+1} = g_s(W_{ss}s_t + W_{sa}a_t + b_s)$$

$$y_t = g_o(W_{os}s_t + b_o)$$

$W_{ss} \in \mathbb{R}^{n_s \times n_s}$, $W_{sa} \in \mathbb{R}^{n_s \times n_a}$, $b_s \in \mathbb{R}^{n_s}$

$W_{os} \in \mathbb{R}^{n_o \times n_s}$, $b_o \in \mathbb{R}^{n_o}$.

- Above $g_s : \mathbb{R}^{n_s} \to \mathbb{R}^{n_s}$, $g_o : \mathbb{R}^{n_o} \to \mathbb{R}^{n_o}$ are similar to activation functions in neural networks
  - They are applied coordinate-wise. e.g. $g_s(x) = \tanh(x) \to (\tanh(x_1), \ldots, \tanh(x_n))$
  - The output $g_0(\,.\,)$: often set as id(.), identity function for regression, or sigmoid (2 category classification) and softmax(.) (> 2 categories classification)

- State s_t carries memory. In RNN s_t is called *hidden* state, since it's not observed. What is observed is the inputs and outputs over a time interval.

Recurrent neural network examples

- A discrete-time linear dynamical system corresponds to $g_s : \mathbb{R}^{n_s} \to \mathbb{R}^{n_s}$, $g_o : \mathbb{R}^{n_o} \to \mathbb{R}^{n_o}$ being identity maps

$$s_{t+1} = As_t + Ba_t$$
$$y_t = Cs_t$$

You will see it in your control system course but in continuous time.

- Example: consider $n_s = 1, n_a = 1, A = 1, B = 1, C = 1$. Let $s_0 = 0$. Verify that this RNN is an integrator, namely, the output is the integral of the input.

$$s_{t+1} = s_t + a_t \Rightarrow s_t = s_0 + \sum_{i=0}^{t-1} a_i$$
$$y_t = s_t \qquad y_t = s_0 + \sum_{i=0}^{t-1} a_i$$

Recurrent neural network for time-series prediction

- Given a signal $\{x_\tau\}_{\tau=0}^{t}$, the goal is to predict x_{t+1} . Using RNN, we assume the signal is output of some discrete-time dynamical system, and we use $a_\tau = c(x_\tau), y_\tau = c(x_{\tau+1})$, where $c(\,.\,)$ is an encoding of the signal (see next example), for $\tau = 0,1,\ldots,t$.
- Architecture: we fix the state dimension and the functions g_s, g_o.

$s_{t+1} = g_s(W_{ss}s_t + W_{sa}a_t + b_s)$, hidden state of the recurrent neural network

$\hat{y}_t = g_o(W_{os}s_t + b_o)$, output of the recurrent neural network

- Our goal in training is then to determine the parameters, $\theta = \{W_{ss}, W_{sa}, W_{os}, b_s, b_o\}$.

- Inference: Once the parameters are determined, given input $a_t = x_t$, we can determine the

How are the RNNs trained?

We define a loss function $L(\theta) = \frac{1}{t}\sum_{\tau=1}^{t} l(y_\tau, \hat{y}_\tau)$

- Regression: $l(\,.\,,.\,)$ is mean-square error loss $\quad y \in \mathbb{R}^m$
- Classification: $l(\,.\,,.\,)$ is cross-entropy loss $\quad y \in \{1, 2, \cdots, K\}$
- The loss measures how predicted values $\hat{y}_\tau$ match the observed values y_τ

- Perform gradient descent on the loss: $\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k)$

- The loss function is generally non-convex in the parameters. Hence, we don't have any guarantee of convergence to the optimal parameters.
- As with other neural networks, when the training and test error appear small, we stop

Next character prediction through RNN

- Training text: ‘h e l l o‘
- Goal: predict next character at each time step
- Approach: note that here actions are the inputs to the RNN
  - Character: x_τ , one-hot encoding $a_\tau = \text{one-hot}(x_\tau)$, target: $y_\tau = \text{one-hot}(x_{\tau+1})$
  - Assume alphabet is $A = \{\text{h}, \text{e}, \text{l}, \text{o}\}$ Note: in early language models alphabet includes all letters and punctuation

$$\text{h} \rightarrow a = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{e} \rightarrow a = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{l} \rightarrow a = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \text{o} \rightarrow a = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Next character prediction through RNN, 2

- RNN equations

Picked $n_s \Rightarrow s_t \in \mathbb{R}^{n_s}$, g_s,

$$s_{t+1} = g_s(W_{ss}s_t + W_{sa}a_t + b_s), \text{ with } s_0 = 0_{n_s \times 1}$$

$$\hat{y}_t = \text{softmax}(W_{os}s_t + b_o)$$

- Training objective $L(\theta) = \frac{1}{t}\sum_{\tau=1}^{t} \text{CE}(y_\tau, \hat{y}_\tau)$, where CE is the cross entropy loss
- Model learns statistical dependencies such as $P(\text{l} \mid \text{he}) > P(\text{e} \mid \text{he})$

Beyond RNNs

- RNNs have issues in training due to gradient vanishing/exploding with increasing time (we will see this through an example).
- This has motivated neural networks that improve gradient flow, such as
  - Long Short-Term Memory (LSTM)
  - Transformers: backbone of large language models such as ChatGPT

- While we won't study this in this course, the conceptual mathematical background is the same. We aim to develop a dynamical system model to predict time-series

- Recall: in this course we aim to learn the fundamental mathematical background of ML techniques and see some of their applications

Summary

- Reinforcement learning
  - Policy gradient: an approach to do reinforcement learning

- Recurrent neural network
  - Example of a discrete-time dynamical system
  - Efficiently modeling time-series

- Your tasks this week
  - Problem set 4
  - Work on python homework and exercises