Recap: Long Short-Term Memory

LSTMs
- Inspired by the design of memory cells
- Each module has 4 layers, interacting in a special way.

Recap: Elements of LSTMs

Output gate layer
- Output is a filtered version of our gate state.
- First, apply sigmoid layer to decide what parts of the cell state to output.
- Then, pass the cell state through a tanh (to push the values to be between -1 and 1) and multiply it with the output of the sigmoid gate.

Forget gate layer
- Look at \( h_{t-1} \) and \( x_t \) and output a number between 0 and 1 for each dimension in the cell state \( C_t \).
  - 0: completely delete this,
  - 1: completely keep this.

Update gate layer
- Decide what information to store in the cell state.
- Sigmoid network (input gate layer) decides which values are updated.
- tanh layer creates a vector of new candidate values that could be added to the state.

Recap: Gated Recurrent Units (GRU)

Simpler model than LSTM
- Combines the forget and input gates into a single update gate \( z_t \).
- Similar definition for a reset gate \( r_t \), but with different weights.
- In both cases, merge the cell state and hidden state.

Empirical results
- Both LSTM and GRU can learn much longer-term dependencies than regular RNNs
- GRU performance similar to LSTM (no clear winner yet), but fewer parameters.
Reinforcement Learning

Topics of This Lecture
- Reinforcement Learning
  - Introduction
  - Key Concepts
  - Optimal policies
  - Exploration-exploitation trade-off
- Temporal Difference Learning
  - SARSA
  - Q-Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Reinforcement Learning

- Motivation
  - General purpose framework for decision making.
  - Basis: Agent with the capability to interact with its environment
  - Each action influences the agent’s future state.
  - Success is measured by a scalar reward signal.
  - Goal: select actions to maximize future rewards.

  ![Diagram](image)

- Formalized as a partially observable Markov decision process (POMDP)

The Agent-Environment Interface

- Let’s formalize this
  - Agent and environment interact at discrete time steps \( t = 0, 1, 2, \ldots \)
  - Agent observes state at time \( t: S_t \in S \)
  - Produces an action at time \( t: A_t \in A(S_t) \)
  - Gets a resulting reward \( R_{t+1} \in \mathbb{R} \subset \mathbb{R} \)
  - And a resulting next state: \( S_{t+1} \)

Note about Rewards

- Reward
  - At each time step \( t \), the agent receives a reward \( R_{t+1} \)

- Important note
  - We need to provide those rewards to truly indicate what we want the agent to accomplish.
  - E.g., learning to play chess:
    - The agent should only be rewarded for winning the game.
    - Not for taking the opponent’s pieces or other subgoals.
    - Else, the agent might learn a way to achieve the subgoals without achieving the real goal.

  \( \Rightarrow \) This means, non-zero rewards will typically be very rare!

Reward vs. Return

- Objective of learning
  - We seek to maximize the expected return \( G_t \) as some function of the reward sequence \( R_{t+1}, R_{t+2}, R_{t+3}, \ldots \)
  - Standard choice: expected discounted return

\[
G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
\]

where \( 0 \leq \gamma \leq 1 \) is called the discount rate.

- Difficulty
  - We don’t know which past actions caused the reward.
  \( \Rightarrow \) Temporal credit assignment problem
Markov Decision Processes

- We consider decision processes that fulfill the Markov property.
  - I.e., where the environment’s response at time $r$ depends only on the state and action representation at $t$.

To define an MDP, we need to specify

- State and action sets
- One-step dynamics defined by state transition probabilities
  \[ p(s'|s, a) = \Pr(S_{t+1} = s'| S_t = s, A_t = a) \]
- Expected rewards for next-state-action-next-state triplets
  \[ r(s, a, s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] = \sum_{s''} p(s'' | s', a) \]

Deterministic policy:

- Under a policy $\sigma$ $\forall s \in S$
  \[ \sigma(s) = a \]

Two types of policies

- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \Pr(A_t = a | S_t = s)$

Note

- $\pi(a|s)$ denotes the probability of taking action $a$ when in state $s$.

Value Function

- Idea
  - Value function is a prediction of future reward
  - Used to evaluate the goodness/badness of states
  - And thus to select between actions

Definition

- The value of a state $s$ under a policy $\pi$, denoted $v_\pi(s)$, is the expected return when starting in $s$ and following $\pi$ thereafter.
  \[ v_\pi(s) = \mathbb{E}_\pi [g_t | S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \]
- The value of taking action $a$ in state $s$ under a policy $\pi$, denoted $q_\pi(s, a)$, is the expected return starting from $s$, taking action $a$, and following $\pi$ thereafter.
  \[ q_\pi(s, a) = \mathbb{E}_\pi [g_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \]

Bellman Equation

- Recursive relationship
  - For any policy $\pi$ and any state $s$, the following consistency holds
    \[ v_\pi(s) = \mathbb{E}_\pi [g_t | S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] = \sum_a \pi(a|s) \sum_{s'} p(s' | s, a) \left[ r + \gamma v_\pi(s') \right] \]
  - This is the Bellman equation for $v_\pi(s)$.

Optimal Value Functions

- For finite MDPs, policies can be partially ordered
  - There will always be at least one optimal policy $\pi^*$.
  - The optimal state-value function is defined as
    \[ v^*_\pi(s) = \max_a v_\pi(s, a) \]
  - The optimal action-value function is defined as
    \[ q^*_\pi(s, a) = \max_a q_\pi(s, a) \]
Optimal Value Functions

- Bellman optimality equations
  - For the optimal state-value function \( v \):
    \[
    v(x) = \max_{a \in \mathcal{A}(x)} q_v(x, a) = \max_{a \in \mathcal{A}(x)} \sum_{x'} p(x', r|x, a) [r + \gamma v(x')]
    \]
  - \( v \) is the unique solution to this system of nonlinear equations.
  - For the optimal action-value function \( q_v \):
    \[
    q_v(x, a) = \sum_{x'} p(x', r|x, a) [r + \max_{a'} q_v(x', a')]
    \]
  - \( q_v \) is the unique solution to this system of nonlinear equations.
  - If the dynamics of the environment \( p(x', r|x, a) \) are known, then in principle one can solve those equation systems.

Optimal Policies

- Why optimal state-value functions are useful
  - Any policy that is greedy w.r.t. \( v \) is an optimal policy.
  - Given \( v \), one-step-ahead search produces the long-term optimal results.
  - Given \( q_v \), we do not even have to do one-step-ahead search
    \[
    \pi_v(x) = \arg \max_{a \in \mathcal{A}(x)} q_v(x, a)
    \]
- Challenge
  - Many interesting problems have too many states for solving \( v \).
  - Many Reinforcement Learning methods can be understood as approximately solving the Bellman optimality equations, using actually observed transitions instead of the ideal ones.

Exploration-Exploitation Trade-off

- Example: N-armed bandit problem
  - Suppose we have the choice between \( N \) actions \( a_1, \ldots, a_N \).
  - If we knew their value functions \( q_v(x, a_i) \), it would be trivial to choose the best.
  - However, we only have estimates based on our previous actions and their returns.
  - We can now
    - **Exploit** our current knowledge
      - And choose the greedy action that has the highest value based on our current estimate.
    - **Explore** to gain additional knowledge
      - And choose a non-greedy action to improve our estimate of that action’s value.

Simple Action Selection Strategies

- \( \epsilon \)-greedy
  - Select the greedy action with probability \((1 - \epsilon)\) and a random one in the remaining cases.
  - In the limit, every action will be sampled infinitely often.
  - Probability of selecting the optimal action becomes \( > (1 - \epsilon) \).
  - But: many bad actions are chosen along the way.
- Softmax
  - Choose action \( a_i \) at time \( t \) according to the softmax function
    \[
    \frac{\exp(\beta q_v(x, a_i))/t}{\sum_{a_j} \exp(\beta q_v(x, a_j))/t}
    \]
  where \( \beta \) is a temperature parameter (start high, then lower it).
  - Generalization: replace \( q_v \) by a preference function \( h(a_i) \) that is learned by stochastic gradient ascent ("gradient bandit").

Temporal Difference Learning (TD-Learning)

- Policy evaluation (the prediction problem)
  - For a given policy \( \pi \), compute the state-value function \( v_\pi \).
- One option: Monte-Carlo methods
  - Play through a sequence of actions until a reward is reached, then backpropagate it to the states on the path.
  \[
  V(S_t) = V(S_t) + \alpha [R_t + \gamma V(S_{t+1}) - V(S_t)]
  \]
  Target: the actual return after time \( t \)
- Temporal Difference Learning - TD(\( \lambda \))
  - Directly perform an update using the estimate \( V(S_{t+1}) \).
  \[
  V(S_t) = V(S_t) + \alpha [R_t + \gamma V(S_{t+1}) - V(S_t)]
  \]
  Target: an estimate of the return (here: TD(0))
SARSA: On-Policy TD Control

- **Idea**
  - Turn the TD idea into a control method by always updating the policy to be greedy w.r.t. the current estimate

- **Procedure**
  - Estimate $q_\pi(s,a)$ for the current policy $\pi$ and for all states $s$ and actions $a$.
  - TD(0) update equation
    \[ Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \]
  - This rule is applied after every transition from a nonterminal state $S_t$.
  - It uses every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.
  - Hence, the name SARSA.

---

Q-Learning: Off-Policy TD Control

- **Idea**
  - Directly approximate the optimal action-value function $q_\ast$, independent of the policy being followed.

- **Procedure**
  - TD(0) update equation
    \[ Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \]
  - Dramatically simplifies the analysis of the algorithm.
  - All that is required for correct convergence is that all pairs continue to be updated.

---

Topics of This Lecture

- Reinforcement Learning
  - Introduction
  - Key Concepts
  - Optimal policies
  - Exploration-exploitation trade-off
- Temporal Difference Learning
  - SARSA
  - Q-Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

---

Deep Reinforcement Learning

- RL using deep neural networks to approximate functions
  - Value functions
    - Measure goodness of states or state-action pairs
  - Policies
    - Select next action
  - Dynamics Models
    - Predict next states and rewards

---

Slide credit: Sergey Levine
Deep Reinforcement Learning

- Application: Learning to play Atari games

Input: pixels + game scores
Output: control commands

B. Leibe

Idea Behind the Model

- Interpretation
  - Assume finite number of actions
  - Each number here is a real-valued quantity that represents the Q function in Reinforcement Learning
  - Collect experience dataset:
    - Set of tuples \{(s,a,s',r), \ldots \}
    - (State, Action taken, New state, Reward received)

- L2 Regression Loss

\[
L(\theta) = \E_{(s,a,s',r) \sim D} \left[ (r + \gamma \max_{a'} Q(s',a'; \theta^*) - Q(s,a; \theta))^2 \right]
\]

Slide credit: Andrej Karpathy
B. Leibe

Results: Space Invaders

B. Leibe

Results: Breakout

B. Leibe

Comparison with Human Performance

Close-up view

Learned Representation

- t-SNE embedding of DQN last hidden layer (Space Inv.)
References and Further Reading

• More information on Reinforcement Learning can be found in the following book

  Richard S. Sutton, Andrew G. Barto
  Reinforcement Learning: An Introduction
  MIT Press, 1998

• The complete text is also freely available online