UNIT-III

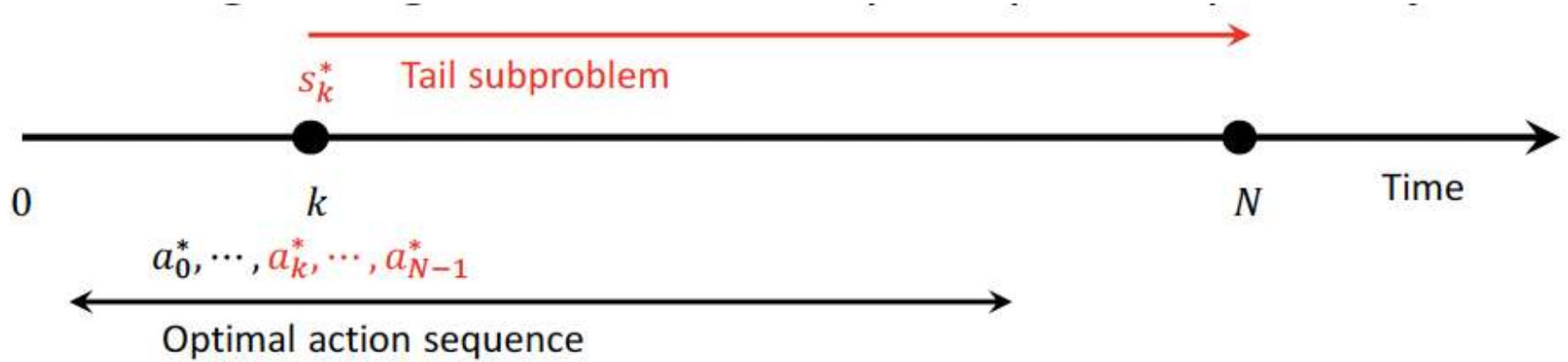
Overview of dynamic programing for MDP, Definition and formulation of planning in MDPs, principle of optimality, Policy Evaluation, Policy Improvement, Policy Iteration, Value Iteration, Generalized Policy Iteration, Efficiency of Dynamic Programming, Banach fixed point theorem, proof of convergence of policy evaluation and value iteration algorithms

Dynamic Programing

- Dynamic Programing [DP] in this course, refer to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment in a MDP.
- Limited utility due to the `perfect model' assumption and due to computational expense.
- But still are important as they provide essential foundation for many of the subsequent methods.
- Many of the methods can be viewed as attempts to achieve much the same effect as DP with less computation and without perfect model assumption of the environment.
- The key idea in DP is to use the value functions and Bellman equations to organize and structure the search for good policies.
- Dynamic Programing addresses a bigger problem by breaking it down as subproblems and then
  - Solving the subproblems
  - Combining solutions to subproblems

Dynamic Programming

- *Dynamic Programing is based on the principle of optimality.*



Requirements for Dynamic Programming

- x Optimal substructure i.e., principle of optimality applies.
- x Overlapping subproblems, i.e., subproblems recur many times and solutions to these subproblems can be cached and reused.
- x MDPs satisfy both through Bellman equations and value functions.
- x Dynamic programming is used to solve many other problems, e.g., Scheduling algorithms, Graph algorithms (e.g. shortest path algorithms), Bioinformatics etc.

Planning by Dynamic Programing

- Planning by dynamic programing assumes full knowledge of the MDP
- For *prediction/evaluation*
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - Output: Value function v_π
- For *control*
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: Optimal value function v_* and optimal policy π_*

Iterative Policy Evaluation

- Problem: Policy evaluation: Compute the state-value function v_π for an arbitrary policy π.
- Solution strategy: Iterative application of Bellman expectation equation.
- Recall the Bellman expectation equation.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_\pi(s') \right\} \tag{1}$$

- Consider a sequence of approximate value functions $v^{(0)}, v^{(1)}, v^{(2)}, \dots$ each mapping $\mathcal{S}^+$ to $\mathbb{R}$. Each successive approximation is obtained by using eqn. (1) as an update rule.

$$v^{(k+1)}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^{(k)}(s') \right\}$$

Iterative Policy Evaluation

$$v^{(k+1)}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^{(k)}(s') \right\}$$

- In code, this can be implemented by using two arrays - one for the old values $v^{(k)}(s)$ and the other for the new values $v^{(k+1)}(s)$. Here, new values of $v^{(k+1)}(s)$ are computed one by one from the old values $v^{(k)}(s)$ without changing the old values.
- Another way is to use one array and update the values 'in place', *i.e.*, each new value immediately overwriting the old one.
- Both these converges to the true value v_π and the 'in place' algorithm usually converges faster.

Iterative Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input: π, the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0|$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$$V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right\}$$

$\Delta \leftarrow \max\left(\Delta, |v - V(s)|\right)$

until $\Delta < \theta$

Evaluating a Random Policy in the Small Gridworld

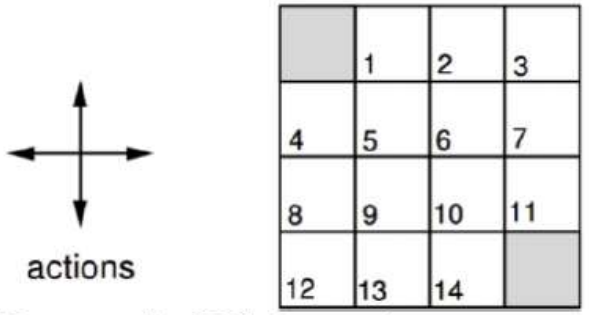
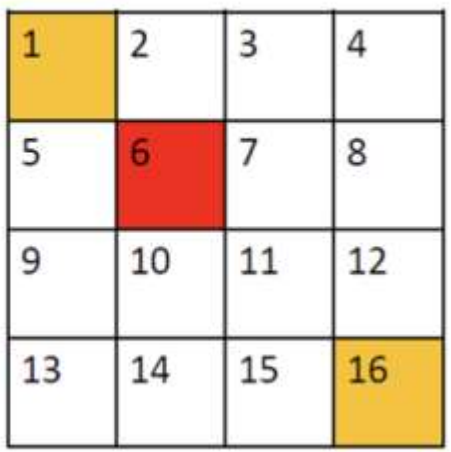


Figure credit: *[SB] chapter 4*

- Undiscounted episodic MDP $(\lambda = 1)$
- Non-terminal states are $\mathcal{S} = \{1, 2, \cdots, 14\}$
- Two terminal states (shown as shaded squares)
- 4 possible actions in each state, $\mathcal{A} = \{up, down, right, left\}$
- Deterministic state transitions
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy
  $\pi(n|.) = \pi(s|.) = \pi(e|.) = \pi(w|.)$

We will start with initialising v_0 for the random policy to all 0s.

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



$$v_1(6) = \sum_{a\in\{u,d,l,r\}} \pi(a|6) \sum_{s',r} p(s',r|6,a)[r + \gamma v_0(s')]$$

$$= \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{=0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_0(s')}_{=0\ \forall s'}]$$

$$= 0.25 * \{-p(2|6,u) - p(10|6,d) - p(5|6,l) - p(7|6,r)\}$$

$$= 0.25 * \{-1 - 1 - 1 - 1\}$$

$$= -1$$

$$\Rightarrow v_1(6) = -1$$

For terminal states p(s'/s,a) = 0 and hence $v_k(1) = v_k(16) = 0$ for all k. So v_1 for the random policy is given by:

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

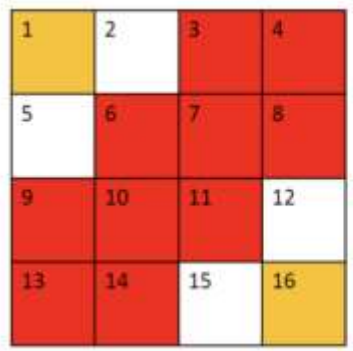
Now, for $v_2(s)$ we are assuming γ or the discounting factor to be 1:

$$v_2(6) = \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{=0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_1(s')}_{=\begin{cases}-1, s'\in S\\ 0, s'\in S^+\setminus S\end{cases}}]$$

$$= 0.25 * \{p(2|6,u)[-1-\gamma] + p(10|6,d)[-1-\gamma] + p(5|6,l)[-1-\gamma] + p(7|6,r)[-1-\gamma]\}$$

$$\overset{\gamma=1}{=} 0.25 * \{-2-2-2-2\}$$

$$= -2$$



For all the remaining states, i.e., 2, 5, 12 and 15, v_2 can be calculated as follows:

$$v_2(2) = \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|2)}_{=0.25\ \forall a} \sum_{s'} p(s'|2,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_1(s')}_{=\begin{cases}-1, s'\in S\\ 0, s'\in S^+\setminus S\end{cases}}]$$

$$= 0.25 * \{p(2|2,u)[-1-\gamma] + p(6|2,d)[-1-\gamma] + p(1|2,l)[-1-\gamma*0] + p(3|2,r)[-1-\gamma]\}$$

$\gamma = 1$

$$= 0.25 * \{-2-2-1-2\}$$

$$= -1.75$$

$$\Rightarrow v_2(2) = -1.75$$

$\Rightarrow v_2$ for the random policy:

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

If we repeat this step several times, we get v_π:

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$\cdots$

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$\cdots$

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\leftarrow v_\pi$

Improving a Policy: Policy Iteration

- Given a policy π
  - Evaluate the policy

$$v_\pi \doteq v^{(k+1)}(s) \leftarrow \sum_{a\in\mathcal{A}} \pi(a|s) \left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v^{(k)}(s') \right\}$$

  - Improve the policy by acting greedily with respect to v_π

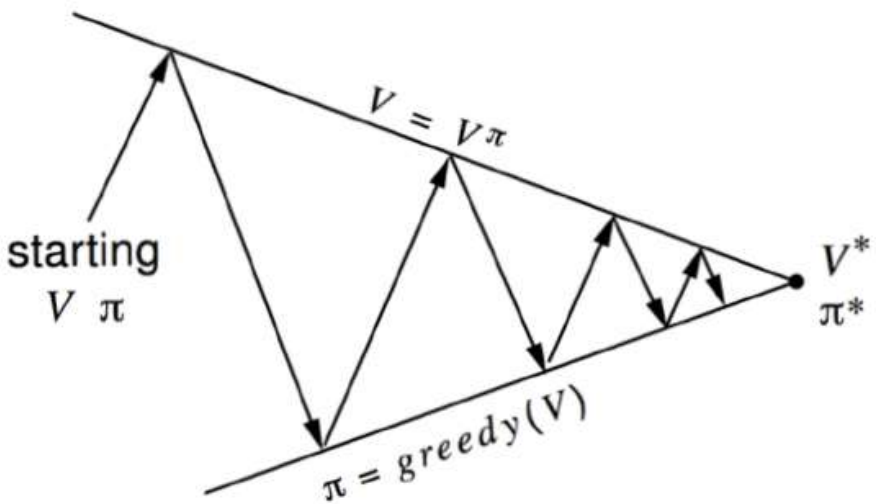
$$\pi' = \text{greedy}(v_\pi)$$

being greedy means choosing the action that will land the agent into best state *i.e.*, $\pi'(s) \doteq \arg\max_{a\in\mathcal{A}} q_\pi(s,a) =$

$$\arg\max_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v_\pi(s') \right\}$$

- In Small Gridworld improved policy was optimal $\pi' = \pi_*$
- In general, need more iterations of improvement/evaluation
- But this process of policy iteration always converges to π_*

Policy Iteration



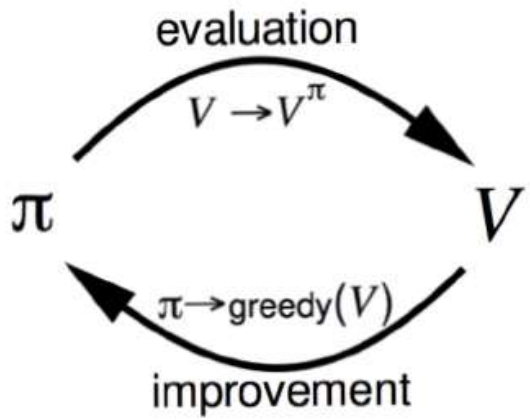


Figure credit: *[David Silver: DeepMind]*

- Policy Evaluation: Estimate v_π by iterative policy evaluation.
- Policy Improvement: Generate $\pi' \geq \pi$ by greedy policy improvement.

Policy Iteration

Algorithm 1: Policy iteration

initialization: Select π^0, $n \leftarrow 0$;

do

  (Policy Evaluation) $v_{(\pi^{n+1})} \leftarrow r_{(\pi^n)} + \gamma \mathcal{P}_{\pi^n} v_{(\pi^n)}$; // componentwise

  (Policy Improvement)

$$\pi^{n+1}(s) \in \arg\max_{a \in \mathcal{A}} [r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_{(\pi^{n+1})}(s')] \; \forall s \in \mathcal{S}];$$

  $n \leftarrow n + 1$;

while $\pi^{n+1} \neq \pi^n$;

Declare $\pi^* = \pi^n$

Policy Iteration: Disadvantages

- Policy iteration involves the policy evaluation step first and this itself requires a few iterations to get the exact value of v_π in limit.
- The question is - must we wait for exact convegence to v_π? Or can we stop short of that?
- The small gridworld example showed that there is no change of the greedy policy after the first three iterations.
- So the question is - is there such a number of iterations such that after that the greedy policy does not change?

Value Iteration

- A related question is - what about the extreme case of 1 iteration of policy evaluation and then greedy policy improvement? If we repeat this cycle, does it find the optimal policy at least in limit?
- The good news is that - yes the gurantee is there and we will soon prove that. However, first let us modify the policy iteration algorithm to this extreme case. This is known as 'value iteration' strategy.

Value Iteration

- What policy iteration does: iterate over

$$v_\pi \doteq v^{(k+1)}(s) \leftarrow \sum_{a\in\mathcal{A}} \pi(a|s) \left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v^{(k)}(s') \right\}$$

- And then

$$\pi'(s) = \arg\max_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v_\pi(s') \right\}$$

- What value iteration does: evaluate $\forall a \in \mathcal{A}$

$$r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v^{(k)}(s')$$

- And then take max over it

$$v^{(k+1)}(s) = \max_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v^{(k)}(s') \right\}$$ Where have we seen it?

Value Iteration

Algorithm 2: Value iteration

initialization: $v \leftarrow v^0 \in \mathcal{V}$, pick an $\epsilon > 0$, $n \leftarrow 0$;

while $||v^{n+1} - v^n|| > \epsilon \frac{1-\gamma}{2\gamma}$ **do**

  foreach $s \in \mathcal{S}$ **do**

    $v^{n+1}(s) \leftarrow \max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'/s,a) v^n(s') \right\}$

  end

  $n \leftarrow n + 1$;

end

foreach $s \in \mathcal{S}$ **do**

  /* Note the use of $\pi(s)$. It mens deterministic policy */

  $\pi(s) \leftarrow \arg\max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'/s,a) v^n(s') \right\}$; // n has already been incremented by 1

end

Summary of Exact DP Algorithms for Planning

Problem	**Bellman Equation**	**Algorithm**
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Norms

Definition

Given a vector space $\mathcal{V} \subseteq \mathbb{R}^d$, a function $f : \mathcal{V} \to \mathbb{R}^+$ is a norm (denoted as $||.||$) if and only if

- $||v|| \geq 0 \quad \forall v \in \mathcal{V}$
- $||v|| = 0$ if and only if $v = 0$
- $||\alpha v|| = |\alpha| \, ||v||$, $\forall \alpha \in \mathbb{R}$ and $\forall v \in \mathcal{V}$
- **Triangle inequality:** $||u + v|| \leq ||u|| + ||v|| \quad u, v \in \mathcal{V}$

Different types of Norms

- L_p norm:

$$||v||_p = \left(\sum_{i=1}^{d} |v_i|^p \right)^{\frac{1}{p}}$$

- L_0 norm:

$$||v||_0 = \text{Number of non-zero elements in } v$$

- L_∞ norm:

$$||v||_\infty = \max_{1 \leq i \leq d} |v_i|$$

Cauchy Sequence, Completeness

Definition

A sequence of vectors $v_1, v_2, v_3, \cdots \in \mathcal{V}$ (with subscripts $n \in \mathbb{N}$) is called a **Cauchy sequence** if for any positive real $\epsilon > 0, \exists\ N \in \mathbb{Z}^+$ such that $\forall m, n > N,\ ||v_m - v_n|| < \epsilon$.

- Basically, for any real positive ϵ, an element can be found in the sequence, beyond which any two elements of the sequence will be within ϵ of each other.
- In other words, the elements of the sequence comes closer and closer to each other - *i.e.*, the sequence converges.

Definition

A vector space $\mathcal{V}$ equipped with a norm $||.||$ is **complete** if **every** Cauchy sequence converges in that norm to a point **in the space**. To pay tribute to Stefan Banach, the great Polish mathematician, such a space is also called the **Banach space**.

Contraction Mapping, Fixed Point

Definition

An operator $\mathcal{T} : \mathcal{V} \to \mathcal{V}$ is **L-Lipschitz** if for any $u, v \in \mathcal{V}$

$$||\mathcal{T}u - \mathcal{T}v|| \leq L||u - v||$$

- If $L \leq 1$, then $\mathcal{T}$ is called a **non-expansion**, while if $0 \leq L < 1$, then $\mathcal{T}$ is called a **contraction**.

Definition

Let v is a vector in the vector space $\mathcal{V}$ and $\mathcal{T}$ is an operator $\mathcal{T} : \mathcal{V} \to \mathcal{V}$. Then v is called a **fixed point** of the operator $\mathcal{T}$, if $\mathcal{T}v = v$.

Banach Fixed Point Theorem

Theorem

Suppose $\mathcal{V}$ is a Banach space and $T : \mathcal{V} \to \mathcal{V}$ is a contraction mapping, then,
- $\exists$ an unique v^* in $\mathcal{V}$ s.t. $Tv^* = v^*$ and
- for arbitrary v^0 in $\mathcal{V}$, the sequence $\{v^n\}$ defined by $v^{n+1} = Tv^n = T^{n+1}v^0$, converges to v^*.

The above theorem tells that

- T has fixed point, **an unique fixed point**.
- For **arbitrary** starting point if we keep repeatedly applying T on that starting point, then we will converge to v^*.

Banach Fixed Point Theorem - Proof (1)

§ Let v^n and v^{m+n} be two values of v obtained after the n^{th} and the $(n+m)^{th}$ iteration.

$$||v^{m+n} - v^n|| \leq \sum_{k=0}^{m-1} ||v^{n+k+1} - v^{n+k}|| \quad \text{[Triangle inequality]}$$

$$= \sum_{k=0}^{m-1} ||T^{n+k}v^1 - T^{n+k}v^0|| \leq \sum_{k=0}^{m-1} \lambda ||T^{n+k-1}v^1 - T^{n+k-1}v^0||$$

$$\leq \sum_{k=0}^{m-1} \lambda^{n+k} ||v^1 - v^0|| \quad \text{[Repeated use of contraction]}$$

$$= ||v^1 - v^0|| \sum_{k=0}^{m-1} \lambda^{n+k}$$

$$= \frac{\lambda^n(1-\lambda^m)}{1-\lambda} ||v^1 - v^0|| \tag{3}$$

Banach Fixed Point Theorem - Proof (2)

- As m and $n \to \infty$ and as $\lambda < 1$, the norm of difference between v^{m+n} and v^n becomes less and less.
- That means the sequence $\{v^n\}$ is Cauchy.
- And since $\mathcal{V}$ is a Banach space and since every Cauchy sequence converges to a point in that Banach space, therefore the Cauchy sequence $\{v^n\}$ also converges to a point in $\mathcal{V}$.
- What we have proved till now is that the sequence $\{v^n\}$ will reach a converging point in the same space.
- Lets say that the converging point is v^*.
- What we will try to prove next is that v^* is a fixed point and then we will try to prove that v^* is an **unique** fixed point.

Banach Fixed Point Theorem - Proof (3)

- Let us try to see what we get as the norm of the difference between v^* and Tv^*.
- In the first line below we apply triangle inequality where v^n is the value of v at the n^{th} iteration.

$$\begin{aligned} ||Tv^* - v^*|| &\leq ||Tv^* - v^n|| + ||v^n - v^*|| \\ &= ||Tv^* - Tv^{n-1}|| + ||v^n - v^*|| \\ &\leq \lambda||v^* - v^{n-1}|| + ||v^n - v^*|| \text{ [Contraction property]} \end{aligned} \tag{4}$$

- Since $\{v^n\}$ is Cauchy and v^* is the convergence point, both the terms in the above equation will tend to 0 as $n \to \infty$.
- So, as $n \to \infty, ||Tv^* - v^*|| \to 0$. That means in limit $Tv^* = v^*$. So, it is proved that v^* is a fixed point.

Banach Fixed Point Theorem - Proof (4)

- Now we will show the uniqueness, *i.e.*, v^* is unique.
- Let u^* and v^* be two fixed points of the space. From the contraction property, we can write $||Tu^* - Tv^*|| \leq \lambda||u^* - v^*||$.
- But, since u^* and v^* are fixed points, $Tu^* = u^*$ and $Tv^* = v^*$.
- That means $||u^* - v^*|| \leq \lambda||u^* - v^*||$ which can not be true for $\lambda < 1$ unless $v^* = u^*$.
- So, it is proved that v^* is an unique fixed point.