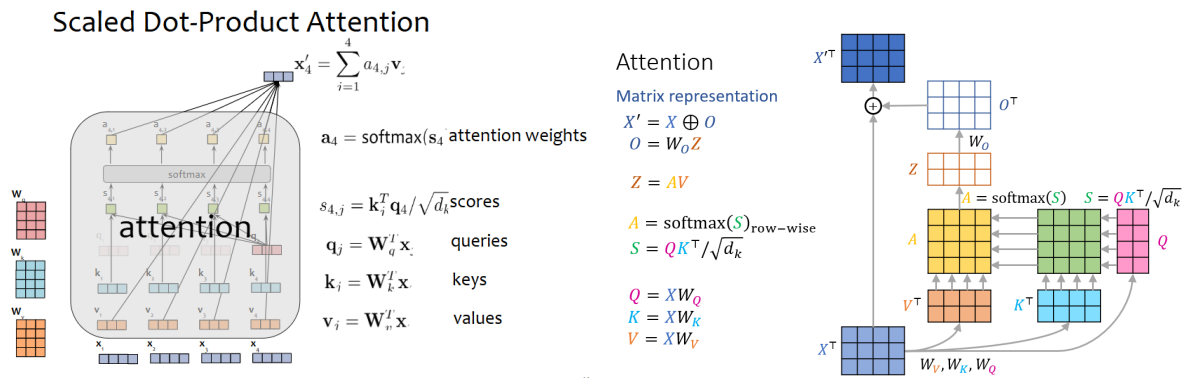


1 Definitions

1.1 Attention

Attention allows neural networks to focus on specific words of an input sequence when generating the output sequence. This means that the importance of different parts of the input are weighed dynamically.

1. Scaled Dot Product Attention: Computes attention scores using the formula $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, where Q (query), K (key), and V (value) are matrices derived from the input, and d_k is the dimension of the keys.



2 MinGPT

MinGPT is a simple implementation of the GPT architecture. In following notebooks, pico version and femto version, we can explore how this model works. For some context, GPT utilizes transformers. A transformer is a type of neural network architecture that focuses on processing sequential data by emphasizing relationships between different parts of the sequence. It achieves this by employing a mechanism called “self-attention,” which, in the context of large language models, allows the model to weigh the importance of each word in the prompt (context) when computing representations.

1. Look at the model architect of either pico or femto version of minGPT. Do you see anything familiar?

Yes. Notice how there are word embedding layers at the beginning of the model. The model also features various linear layers.

2. Compare minGPT pico and minGPT femto architectures. How many attention heads are their in each model? What are the dimension of the word embeddings?

Note: Having multiple attention heads occurs when the attention mechanism is replicated several times in parallel, each time with different learned parameters. Think of it like having multiple channels in convolutions.

Femto is the simpler model. It uses 1 attention head. It has 1 layer and an embedding size of 2.

On the other hand, Pico has 3 attention heads. It has 3 layers, and an embedding size of 6.

3. Now it's time to play around with the model. First, run the untrained pico model on the given context, "I do not like." Generate a sample from the model. What does the model predict?

The output reads, "I do not like fox say with say Let dark , eggs fox do Not , fox . mouse
eggs fox Sam say Let anywhere Eat fox Here fox fox Sam do with with with with with with
with with with with with with with with with with with with with with with".

The model prediction is pretty bad, as expected from an untrained model!

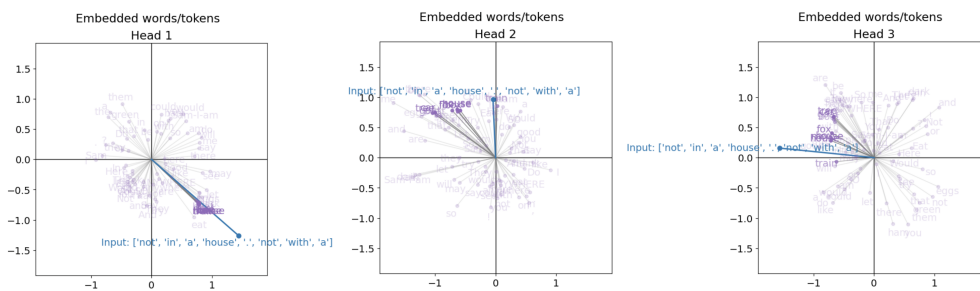
4. Now, train the model for 100 iterations. Do the word embeddings look the same for the vocab? Does the word embedding look the same for the context?

After training for 100 iterations, the embeddings look very different than the ones with zero training. Both the vocab embeddings and the context embedding have shifted significantly.

- Now, let's use the pretrained model on 10,000 iterations and run it with context = "not with in a house , not with a." Sample from this model. What does the model predict? What do the word embeddings for each attention head look like?

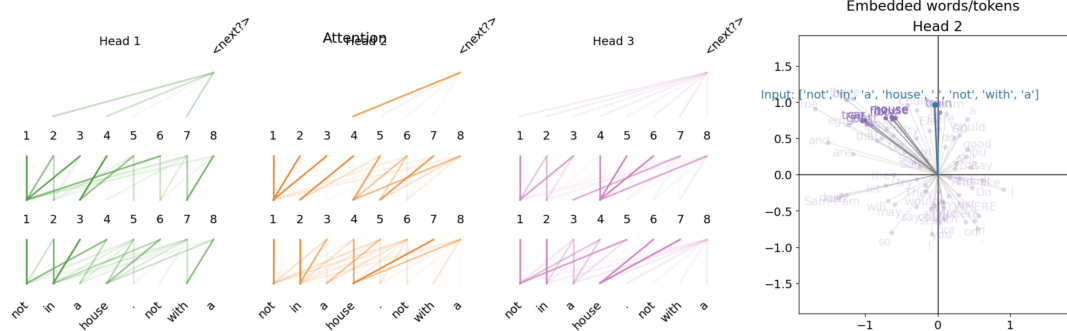
The model outputs, "not with in a house , not with a house . I will not eat them here or there . I will not eat them anywhere . I will not eat them anywhere . I do not like them anywhere . I do not like them , Sam-I-am . Would you , Sam-I-am . Would you , Sam-I-am ."

The word embeddings look pretty good. All three attention heads seem to have found words that would likely come next given the context.



6. For the pretrained model run from the previous part, which attention head appears to be picking up on rhyming?

Head 2



7. Now, compare the results from minGPT pico and minGPT femto. Which model is better?

Pico performs much better. Femto is a very small model and does not appear to be learning very successfully.

Finally, in the notebooks, explore using either model how it generates new text!

3 Pay Attention to Dimensions!

1. Let's say our input sequence is $X \in \mathbb{R}^{T \times d_{model}}$ and our W_K, W_Q, W_V matrices are all in $\mathbb{R}^{d_{model} \times d_k}$.

As a refresher, remember that $V = XW_V$, $Q = XW_Q$, and $K = XW_K$.

- (a) What would be the dimensions of $S = QK^T$?

hw

- (b) How about $A = g_{sm}(S)$? g_{sm} applies the softmax function.

hw

- (c) In the matrix version of attention, this softmax function is applied row or column wise?

see written

2. Let's say that now $d_{model} = 2$, $T = 2$, and $d_k = 2$.

- (a) What would be the dimensions of $S = QK^T$?

hw

- (b) How about $A = g_{sm}(S)$? g_{sm} applies the softmax function.

hw

- (c) How about the W_K, W_Q, W_V matrices?

hw

- (d) How about W, V, Q ?

hw

3. If $T = 3$, which of the following would change dimensions: $W_K, W_V, W_Q, K, V, Q, S, A, X$?

see written

4 MDPs: Warm-Up

1. What is the Markov Property?

The **Markov Property** states that action outcomes depend on the current state only (action outcomes do not depend on the past states and actions).

2. What are the Bellman equations, and when are they used?

The **Bellman Equations** give a definition of “optimal utility” via expectimax recurrence. They give a one-step lookahead relationship between utilities at a current time step and the next time step.

3. What is a policy? What is an optimal policy?

A **policy** is a function that maps states to actions; $\pi(s)$ gives an action for state s .
An **optimal policy** is a policy that maximizes the expected utility if an agent follows it.

4. How does the discount factor γ affect how the agent finds the optimal policy? Why do we restrict gamma $\gamma < 1$?

γ determines how much the value of a state should take into account the value of future states that the agent could wind up in.

A gamma $0 < \gamma < 1$ helps our algorithms converge and to prevents against infinite rewards if our game lasts forever.

5. Fill in the following table explaining the effects of having different gamma values:

γ	Effect on policy search:
$\gamma = 0$	
$\gamma = 1$	

γ	Effect on policy search:
$\gamma = 0$	A smaller γ indicates smaller “horizon,” or a shorter term focus. When γ is 0, we only consider immediate rewards. We do not consider rewards in the future as having any value.
$\gamma = 1$	The higher the discount factor, the more the state would value distant future states. When γ is 1, we begin to act as though rewards at any given point in time are equally valuable.

6. What are the two steps to Policy Iteration?

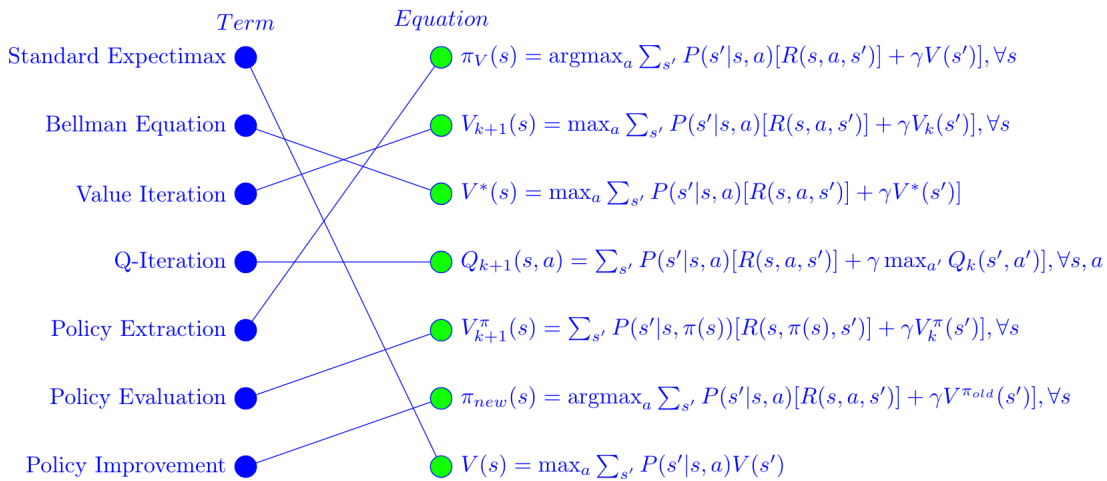
Policy evaluation and policy improvement.

7. What is the relationship between $V^*(s)$ and $Q^*(s, a)$?

$$V^*(s) = \max_a Q^*(s, a)$$

8. (MDP Notation Review) Draw a line connecting each term in the left column with its corresponding equation in the right column.

Term	Equation
Standard Expectimax ●	● $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s' s, a)[R(s, a, s')] + \gamma V(s')], \forall s$
Bellman Equation ●	● $V_{k+1}(s) = \max_a \sum_{s'} P(s' s, a)[R(s, a, s')] + \gamma V_k(s')], \forall s$
Value Iteration ●	● $V^*(s) = \max_a \sum_{s'} P(s' s, a)[R(s, a, s')] + \gamma V^*(s')]$
Q-Iteration ●	● $Q_{k+1}(s, a) = \sum_{s'} P(s' s, a)[R(s, a, s')] + \gamma \max_{a'} Q_k(s', a')], \forall s, a$
Policy Extraction ●	● $V_{k+1}^\pi(s) = \sum_{s'} P(s' s, \pi(s))[R(s, \pi(s), s')] + \gamma V_k^\pi(s')], \forall s$
Policy Evaluation ●	● $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s' s, a)[R(s, a, s')] + \gamma V^{\pi_{old}}(s')], \forall s$
Policy Improvement ●	● $V(s) = \max_a \sum_{s'} P(s' s, a)V(s')$

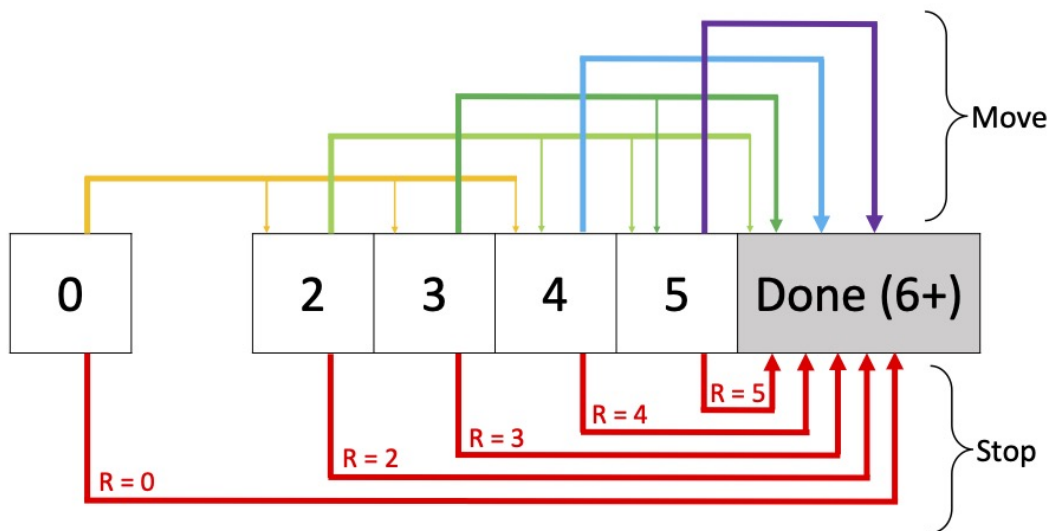


5 MDPs: Racing

Consider a modification of the racing robot car example seen in lecture. In this game, the car repeatedly moves a random number of spaces that is equally likely to be 2, 3, or 4. The car can either Move or Stop if the total number of spaces moved is less than 6.

If the total spaces moved is 6 or higher, the game automatically ends, and the car receives a reward of 0. When the car Stops, the reward is equal to the total spaces moved (up to 5), and the game ends. There is no reward for the Move action.

Let's formulate this problem as an MDP with the states $\{0, 2, 3, 4, 5, Done\}$.



1. What is the transition function for this MDP? (You should specify discrete values for specific state/action inputs.)

$$\begin{aligned}
 T(s, Stop, Done) &= 1, \text{ for } s \neq Done \\
 T(0, Move, s') &= \frac{1}{3} \text{ for } s' \in \{2, 3, 4\} \\
 T(2, Move, s') &= \frac{1}{3} \text{ for } s' \in \{4, 5, Done\} \\
 T(3, Move, 5) &= \frac{1}{3} \\
 T(3, Move, Done) &= \frac{2}{3} \\
 T(4, Move, Done) &= 1 \\
 T(5, Move, Done) &= 1 \\
 T(s, a, s') &= 0 \text{ otherwise.}
 \end{aligned}$$

2. What is the reward function for this MDP?

$$\begin{aligned}
 R(s, Stop, Done) &= s, s \leq 5 \\
 R(s, a, s') &= 0 \text{ otherwise}
 \end{aligned}$$

3. Recall the value iteration update equation:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

Perform value iteration for 4 iterations with $\gamma = 1$.

States	0	2	3	4	5	Done
V_0						0
V_1						0
V_2						0
V_3						0
V_4						0

see written

4. You should have noticed that value iteration converged above. What is the optimal policy?

States	0	2	3	4	5
π^*					

see written

5. How would our results change with $\gamma = 0.1$?

see written

6. Now imagine we changed the rules so that moving to spaces after 5 does not immediately end the game, but rather wraps back around to the beginning states (this is the case where the states exist side by side in a loop).

For example if you start in state 5, select the action Move, and move 2 spaces, you would end up in state 1 (this is a new state we would need introduce).

To be clear, the states would now be states $\{0,1,2,3,4,5, Done\}$.

How would this modification change our results? (Assume we again use $\gamma=1$)

see written

6 MDPs: Policy Iteration

Recall the racing MDP from the prior problem. In this game, the car repeatedly moves a random number of spaces that is equally likely to be 2, 3, or 4. The car can either Move or Stop if the total number of spaces moved is less than 6.

If the total spaces moved is 6 or higher, the game automatically ends, and the car receives a reward of 0. When the car Stops, the reward is equal to the total spaces moved (up to 5), and the game ends. There is no reward for the Move action.

States: $\{0, 2, 3, 4, 5, Done\}$

Transition	Reward
$T(s, Stop, Done) = 1, \forall s \neq Done$	$R(s, Stop, Done) = s, \forall s \leq 5$
$T(0, Move, s') = \frac{1}{3}, \forall s' \in \{2, 3, 4\}$	$R(s, a, s') = 0$ otherwise
$T(2, Move, s') = \frac{1}{3}, \forall s' \in \{4, 5, Done\}$	
$T(3, Move, 5) = \frac{1}{3}$	
$T(3, Move, Done) = \frac{2}{3}$	
$T(4, Move, Done) = 1$	
$T(5, Move, Done) = 1$	
$T(s, a, s') = 0$ otherwise	

Now recall the policy evaluation and policy improvement equations, which together make up policy iteration.

$$\text{Policy Evaluation: } V_{k+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

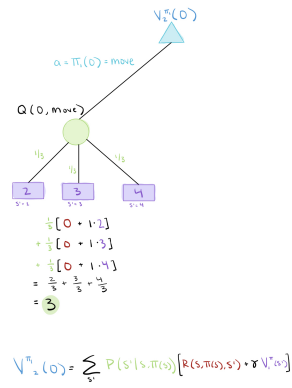
$$\text{Policy Improvement: } \pi_{new}(s) \leftarrow \operatorname{argmax}_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_{old}}(s')]$$

Perform two iterations of policy iteration for one step of this MDP, starting from the fixed policy below. Use the initial $\gamma = 1$.

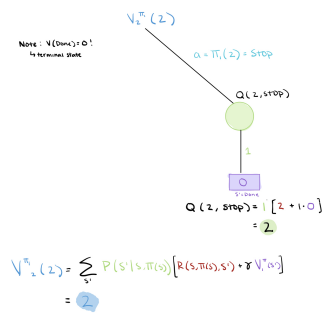
States	0	2	3	4	5
π_0	Move	Stop	Move	Stop	Move
$V_0^{\pi_0}$					
$V_1^{\pi_0}$					
$V_2^{\pi_0}$					
$V_3^{\pi_0}$					
π_1					
$V_0^{\pi_1}$					
$V_1^{\pi_1}$					
$V_2^{\pi_1}$					
$V_3^{\pi_1}$					
π_2					

see written

Policy iteration tree for $V_2^{\pi_1}(0)$:



Policy iteration tree for $V_2^{\pi_1}(2)$:



7 MDPs: Conceptual Questions

1. Which MDP methods are optimal (assuming sufficient convergence when iterating)? [Value Iteration, Policy Iteration, Both, Neither]

Both!

2. What are some limitations of value iteration? What are some limitations of policy iteration?

Limitations of value iteration:

- (a) each iteration takes $O(|S|^2|A|)$ time, which can be costly depending on the size of S and A ;
- (b) values of many states do not change in one iteration, but the process has to continue as long as there is change in some states;
- (c) sometimes the corresponding policy (extract the policy as if the current V_k is V^*) has already converged to optimal, but the values have not converged and therefore we have to continue the value iteration process, which is a waste of time.

Limitation of policy iteration:

- (a) Each iteration of policy iteration involves policy evaluation.
Recall that the equation for policy evaluation is $V_{k+1}^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) [R(s, \pi(s), s')] + \gamma V_k^\pi(s')]$, $\forall s$
Since policy evaluation is an iterative process and is only guaranteed to converge as $k \rightarrow \infty$, we do not know the time of convergence a priori.

Depending on the conditions, one algorithm may converge faster than the other.

3. When does policy iteration end? Immediately after policy iteration ends (without performing additional computation), do we have the values of the optimal policy?

Policy iteration ends when the policy converges, i.e., when $\pi_{new} = \pi_{old}$ after running policy improvement. We do have the values for the optimal policy. Since we necessarily ran policy evaluation on the most recent iteration of policy iteration, we have the value function $V^{\pi_{old}}$ corresponding to π_{old} . We know that $\pi_{new} = \pi_{old}$, implying that $V^{\pi_{old}} = V^{\pi_{new}}$. Therefore, we have $V^{\pi_{new}}$, which is the value function corresponding to the optimal policy π_{new} .

4. What changes if during policy iteration, you only run one iteration of Bellman update instead of running it until convergence? Do you still get an optimal policy?

You will get an optimal policy if the termination condition is updated.

Note: policy iteration can converge too early because the policy might not change between two iterations of changing values. For an example, refer to the value iteration demo in Lecture 15, where we run policy extraction after every iteration of value iteration. There are a couple of early iterations where the non-optimal policy doesn't change.

The key observation here is that policy iteration is effectively the same as value iteration, since value iteration involves one step of evaluation as well. In this case, we update values based on our current best policy, and keep doing that until convergence of a policy!

Recall that the equation for value iteration is:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')] \quad (1)$$

And the equation for policy evaluation is:

$$V_{k+1}^{\pi_i}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^{\pi_i}(s')] \quad (2)$$

given some fixed policy π that we update in the policy improvement step, given by:

$$\pi_{i+1}(s) \leftarrow \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k^{\pi_i}(s')] \quad (3)$$

which looks very similar to policy extraction done in value iteration.