

深層実習 第11,12回

中田尚

課題

☆課題1

- 2つの正方行列の積を求めるpythonプログラムを作成して、行列サイズと実行時間の関係を調べる

☆課題2

- **2つの正方行列の積と正方行列と 3×3 行列の畳み込み**それぞれを求めるpythonプログラムを作成して、それぞれの行列サイズと実行時間の関係を調べる

任意課題

☆課題3

- MNISTの学習において、全結合層と畳み込み層の違いが学習時間と精度に与える影響を調査する

☆課題4

- 簡単なニューラルネットワークを使って、（ハイパー）パラメータの違いが学習時間と精度に与える**影響を調査**する
 - パラメータの例
 - ノード数、層数、活性化関数、epoch数、損失関数
 - 基本的には全結合層と畳み込み層が良いが、それ以外を利用しても良い
 - **注目した**パラメータ（**複数可**）と学習時間と精度のグラフは必須
 - 「**XXを変えても影響がない**」も一つの結果だが、可能な限り「**YYは影響がある**」を含むと良い
 - sklearn.datasetsやtorchvision.datasetsのような「ダウンロードするだけのデータセット」**以外**を使うと+1点（つまり9点満点の上限8点）

深層実習 第11,12回

中田尚

課題4講評

- 当たり前のこと（書いても良いけどほとんど点数にならない）
 - レイヤを増やしたら実行時間が延びた
 - レイヤを増やしたら精度が向上した
 - チャンネル数を増やした等も同様
- 的外れなこと
 - epoch毎の実行時間の変動
 - 精度はあるけど実行時間がない（またはその逆）

トレードオフのある比較

- アルバイト
 - 3時間 5,000円
 - 5時間 5,000円
 - 10時間 12,000円
 - 11時間 10,000円
 - 20時間 100,000円

パレート最適

- 誰かの効用（満足）を犠牲にしなければ、他の誰かの効用を高めることができない状態
- 時間と給料
 - 短時間で多い給料を貰える方が良い
 - 長時間で安いバイトは選ぶ意味が無い
 - 短時間で少ない給料はOK
 - 長時間で多い給料はOK

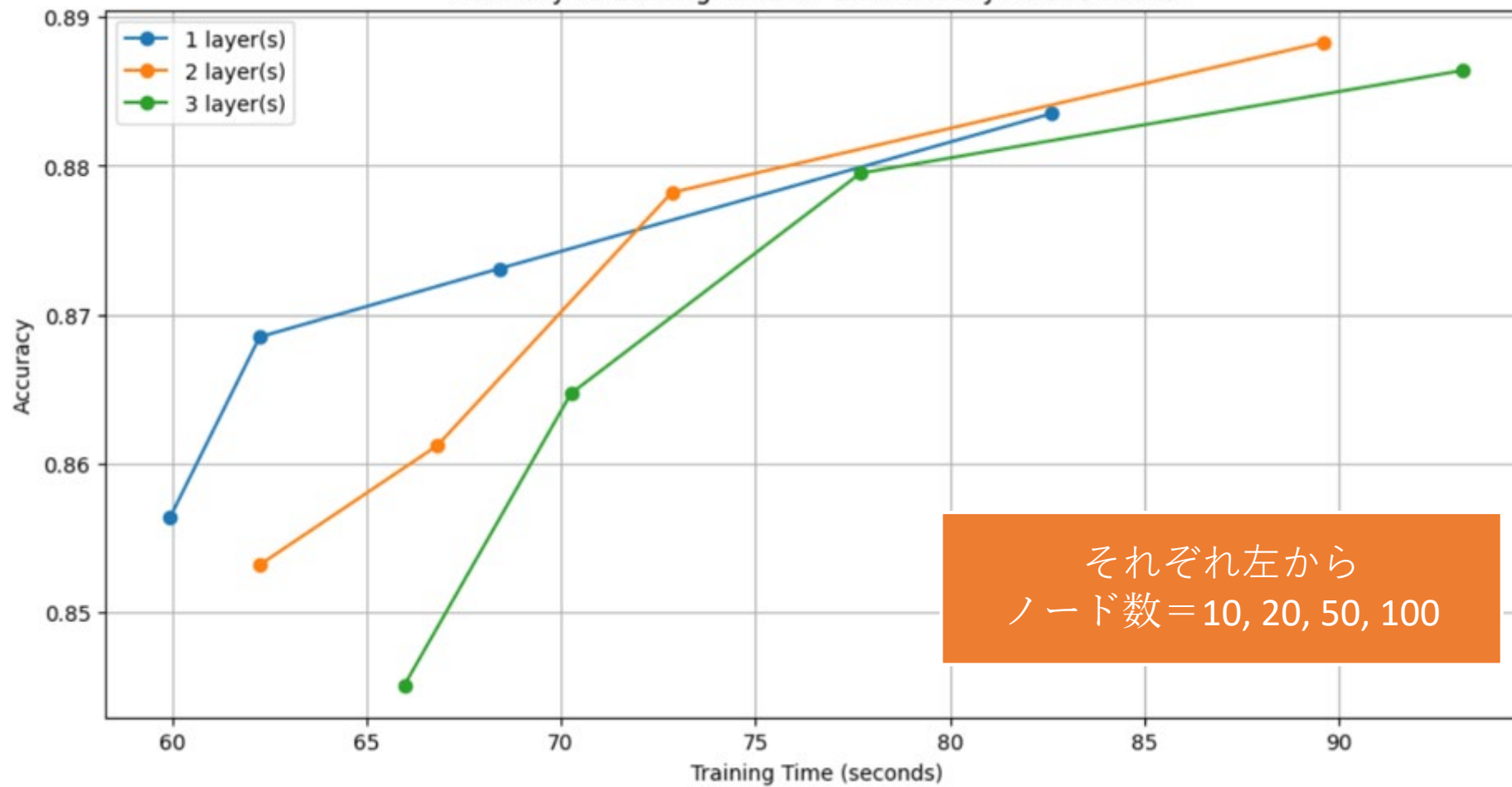
トレードオフのある比較

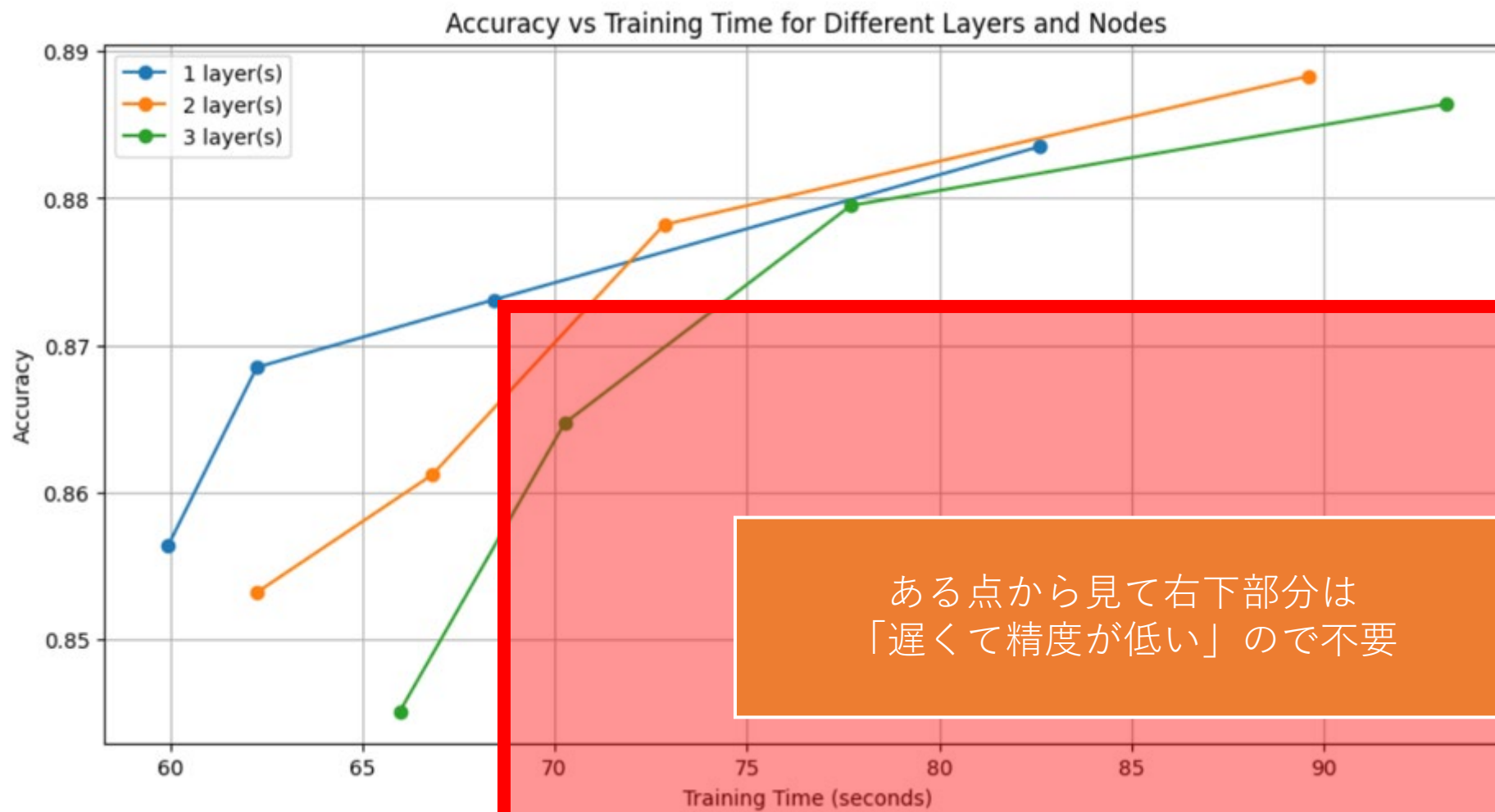
- アルバイト
 - 3時間 5,000円
 - ~~5時間 5,000円~~
 - 10時間 12,000円
 - ~~11時間 10,000円~~
 - 20時間 100,000円

トレードオフのある比較

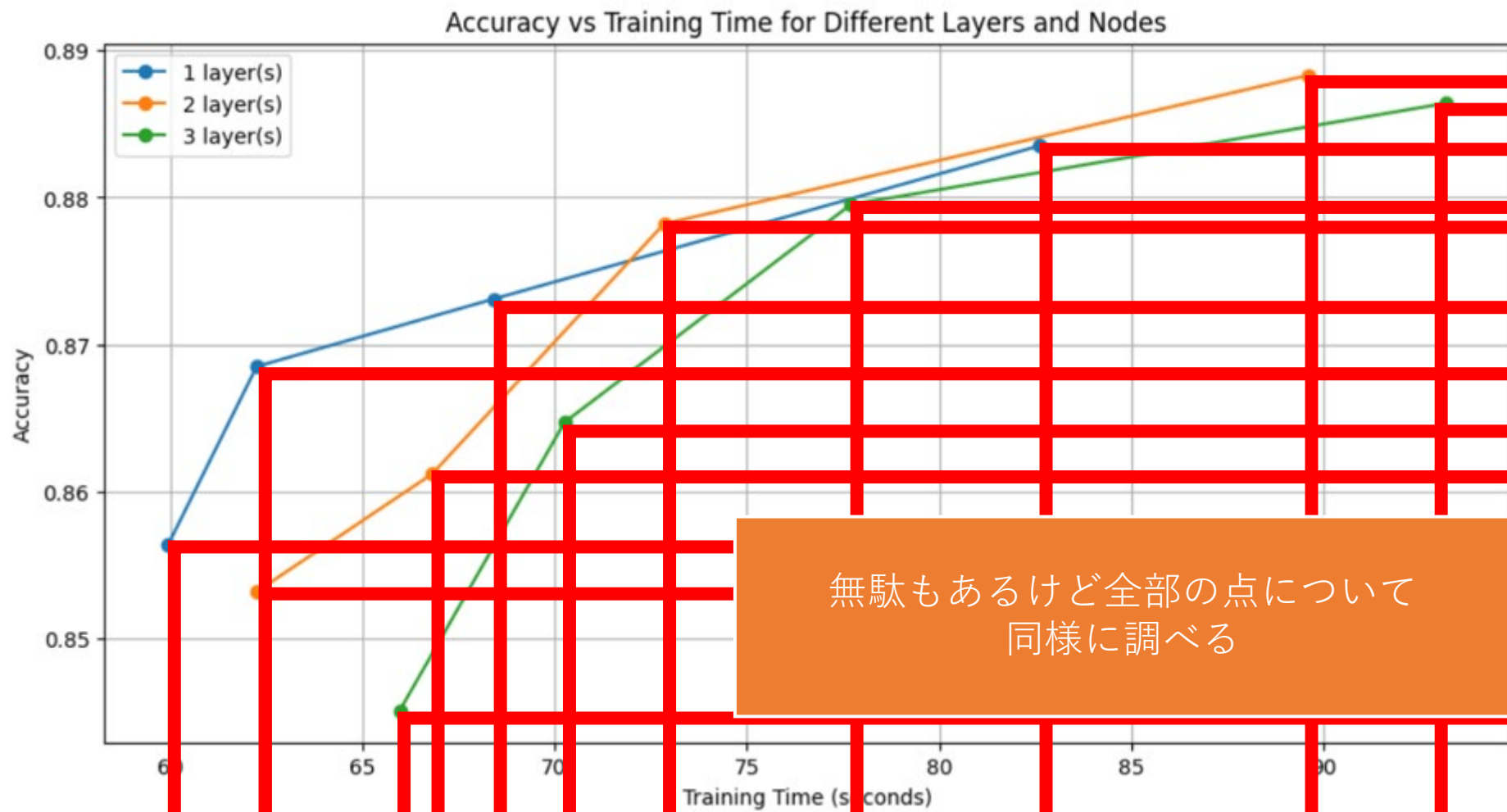
- アルバイト（1日当たり）
 - 3時間 5,000円
 - ~~5時間 5,000円~~
 - 10時間 12,000円
 - ~~11時間 10,000円~~
 - 20時間 100,000円
 - 時給は高いがちょっと難しい

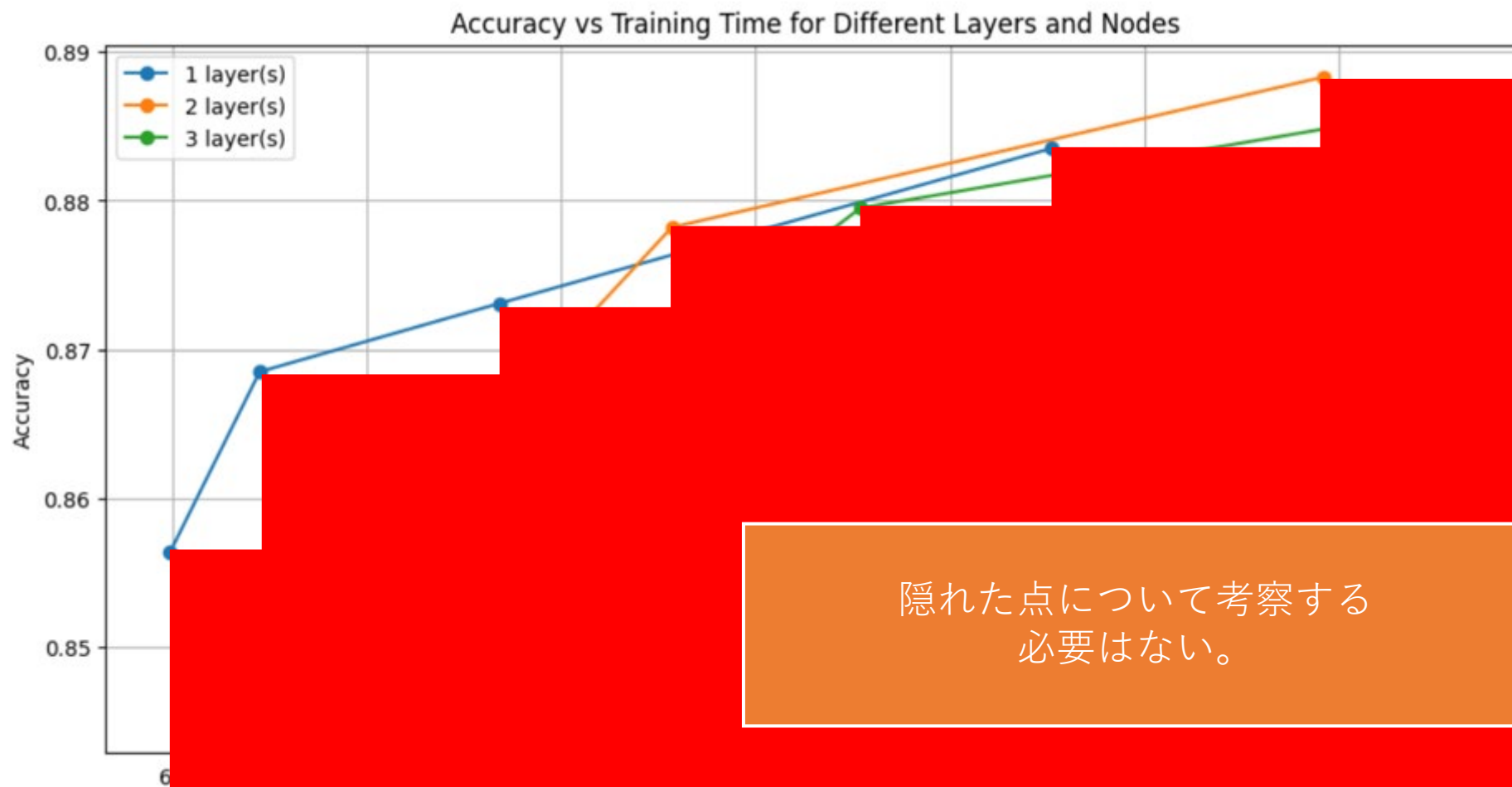
Accuracy vs Training Time for Different Layers and Nodes



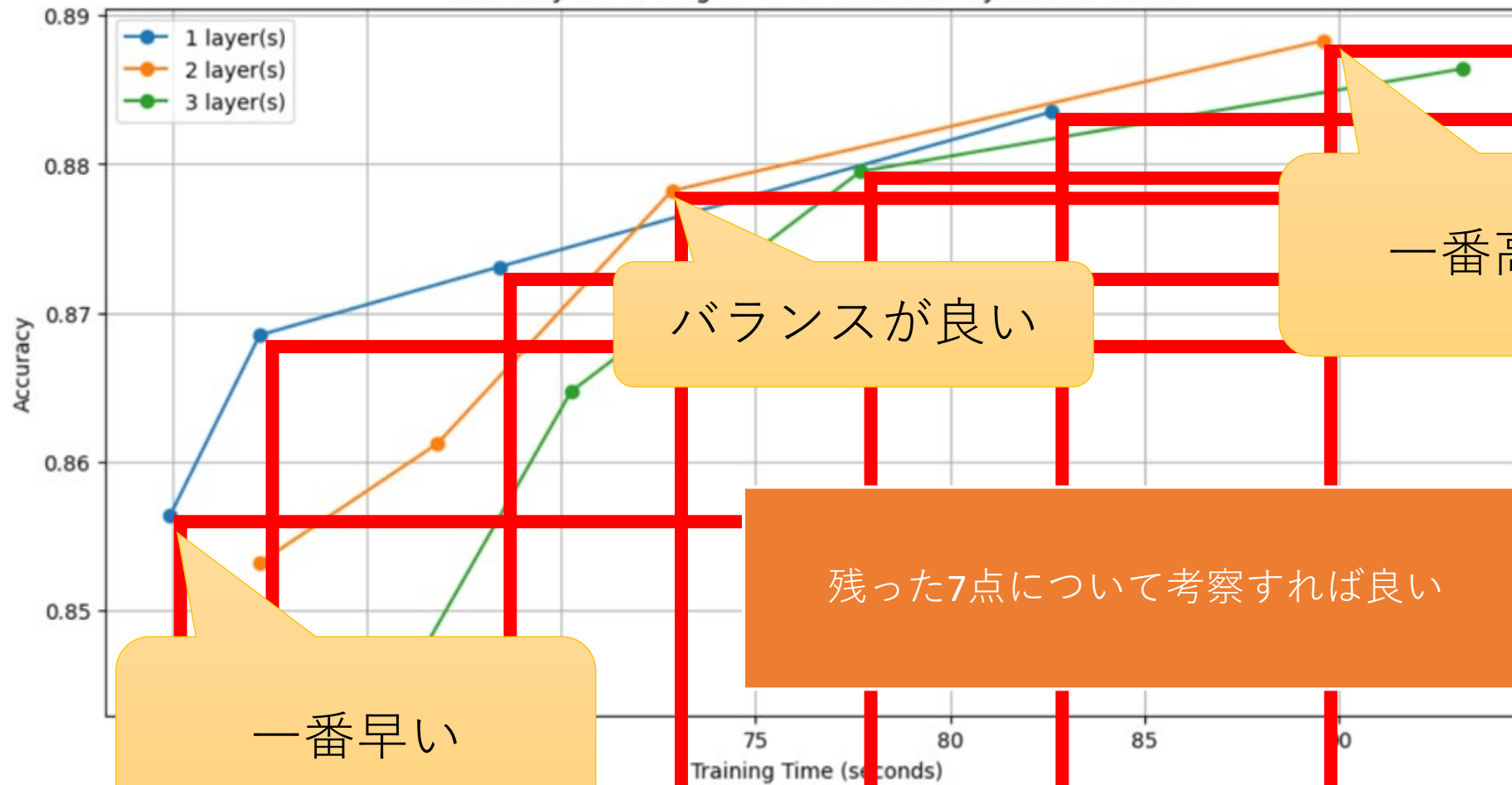


ある点から見て右下部分は
「遅くて精度が低い」ので不要





Accuracy vs Training Time for Different Layers and Nodes



一番早い

バランスが良い

一番高精度

残った7点について考察すれば良い

強化学習

機械学習の代表的な区分

- 「教師あり学習」
 - 入力に対して答えが決まっている
- 「教師なし学習」
 - 入力のみがあり答えがない
 - クラスタリングや次元削減など
- 「強化学習」

強化学習

- 「強化学習」
 - 「状態」
 - 「行動」：選んだ行動により次の状態が決まる
 - 「報酬」：行動の結果報酬が貰えることがある
 - 目的：報酬を最大化する行動を求める
- 「環境」：「状態」＋「行動」＋「報酬」
- 「エージェント」：環境の中で行動し報酬を得る主体

報酬の最大化

- 合計の最大化
 - 例：ひたすら同じ敵を倒して経験値とお金を集める
- 単位時間当たりの合計を最大化
 - 経験値やお金に消費期限があり取得してから一定時間で消えるイメージ
 - レベルを上げて強い敵を倒さなければ、報酬は増加しない
- **指数移動平均**
 - 単位時間で消えるのではなく、徐々に価値が下がっていく
 - n ステップ目の報酬を R_n 、 n ステップ目までの合計を Q_n とすると $Q_n = (1 - \alpha)R_n + \alpha Q_{n-1}$
 - ここで α を割引率と呼ぶ（0.9くらいに設定することが多い）

マルコフ決定過程

「マルコフ性」

- 現在の状態と行動のみから、次の状態と報酬が決まる
 - 過去の行動を考慮する必要はない
 - 過去の行動はすでに現在の状態に反映されている ともいえる
- いわゆる「フラグ」がない

エピソード

- 開始から終了までを1エピソードという
- 終了が無いものを「連続タスク」、終了があるものを「エピソードタスク」という
- 行動回数の上限を定めれば連続タスクもエピソードタスクとして扱える

とりあえず実行

- `state = env.reset()`
- `while True:`
 - `action = agent.get_action(state,best=True)`
 - `next_state, reward, done = env.step(action)`
 - `if done:`
 - `break`
 - `state = next_state`

初期状態を設定

stateにおける最適な
actionを取得

actionの実行

次の状態、報酬、終了判定

次の状態に更新

学習

繰り返し回数

- for episode in range(episode):
- state = env.reset()
- while True:
- action = agent.get_action(state)
- next_state, reward, done = env.step(action)
- agent.update(state, action, reward, next_state, done)
- if done:
- break
- state = next_state

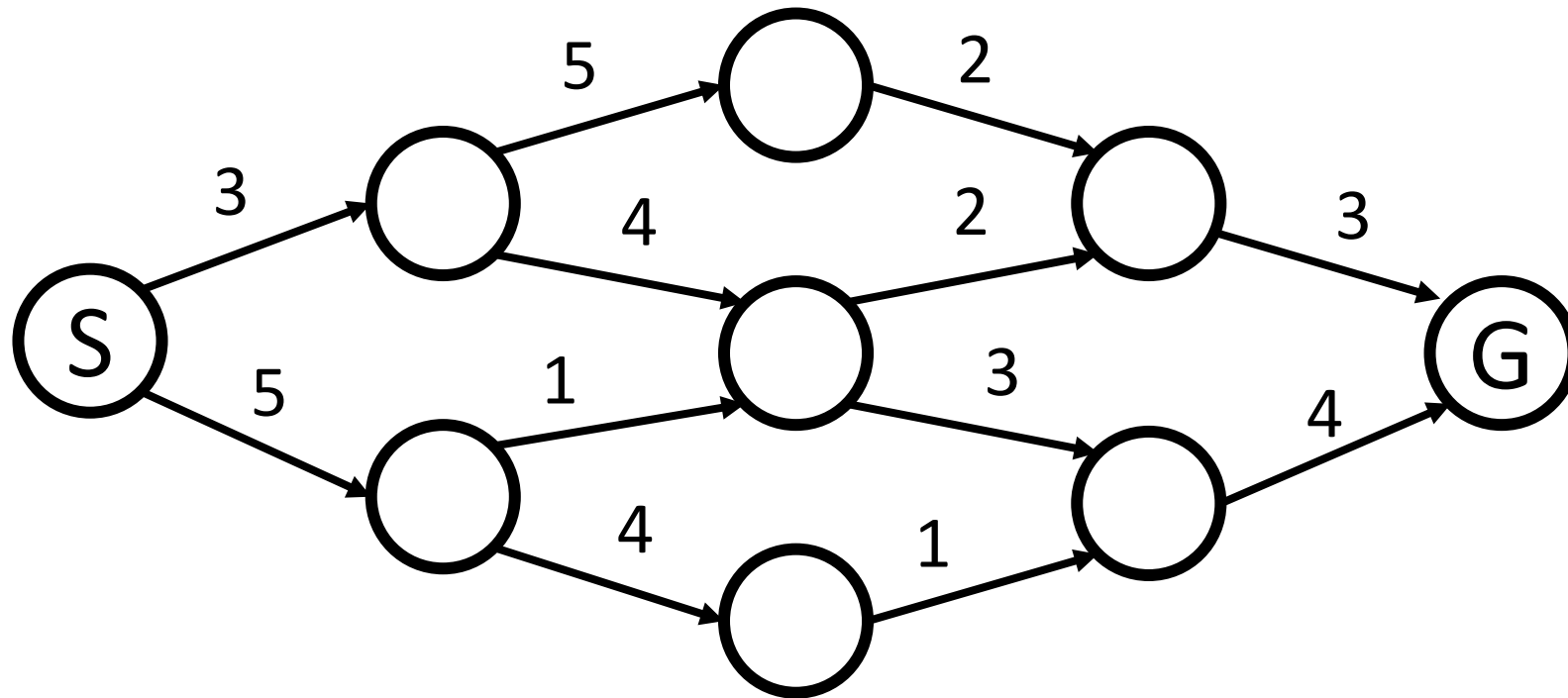
いつなにをしたらどう
なったのかを学習する

Q学習

- 基本的なアイデアは動的計画法

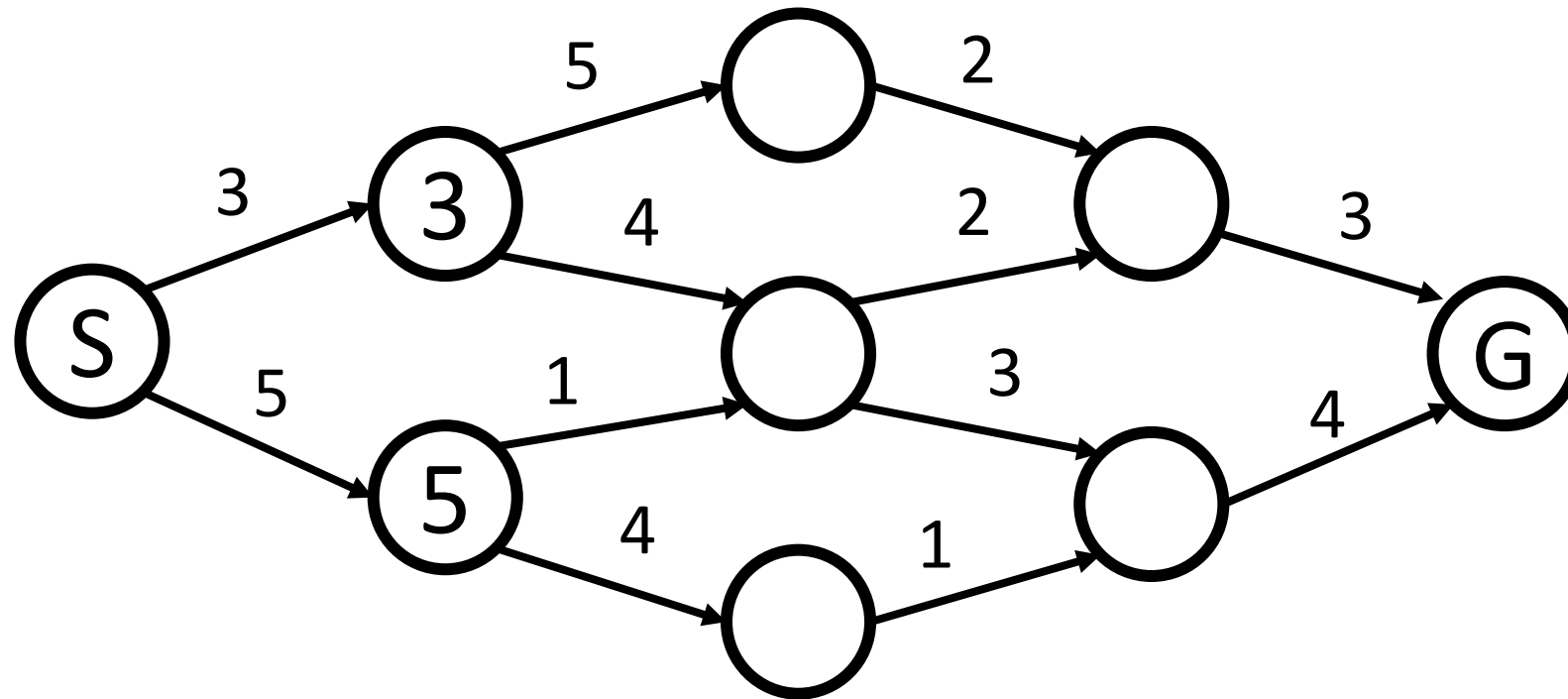
動的計画法の復習

- 数字は移動コスト
- SからGへのコスト最小のルートを探す



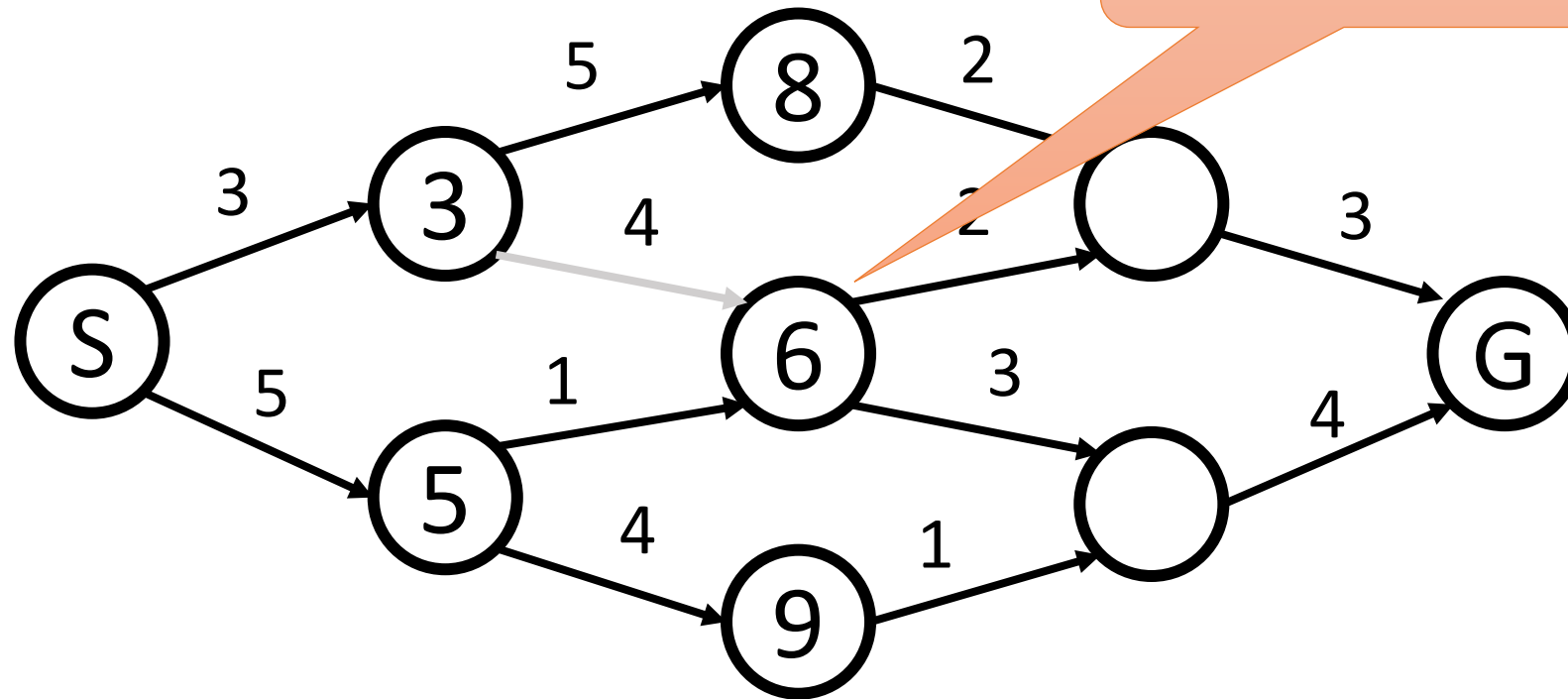
動的計画法の復習

- 数字は移動コスト
- SからGへのコスト最小のルートを探す



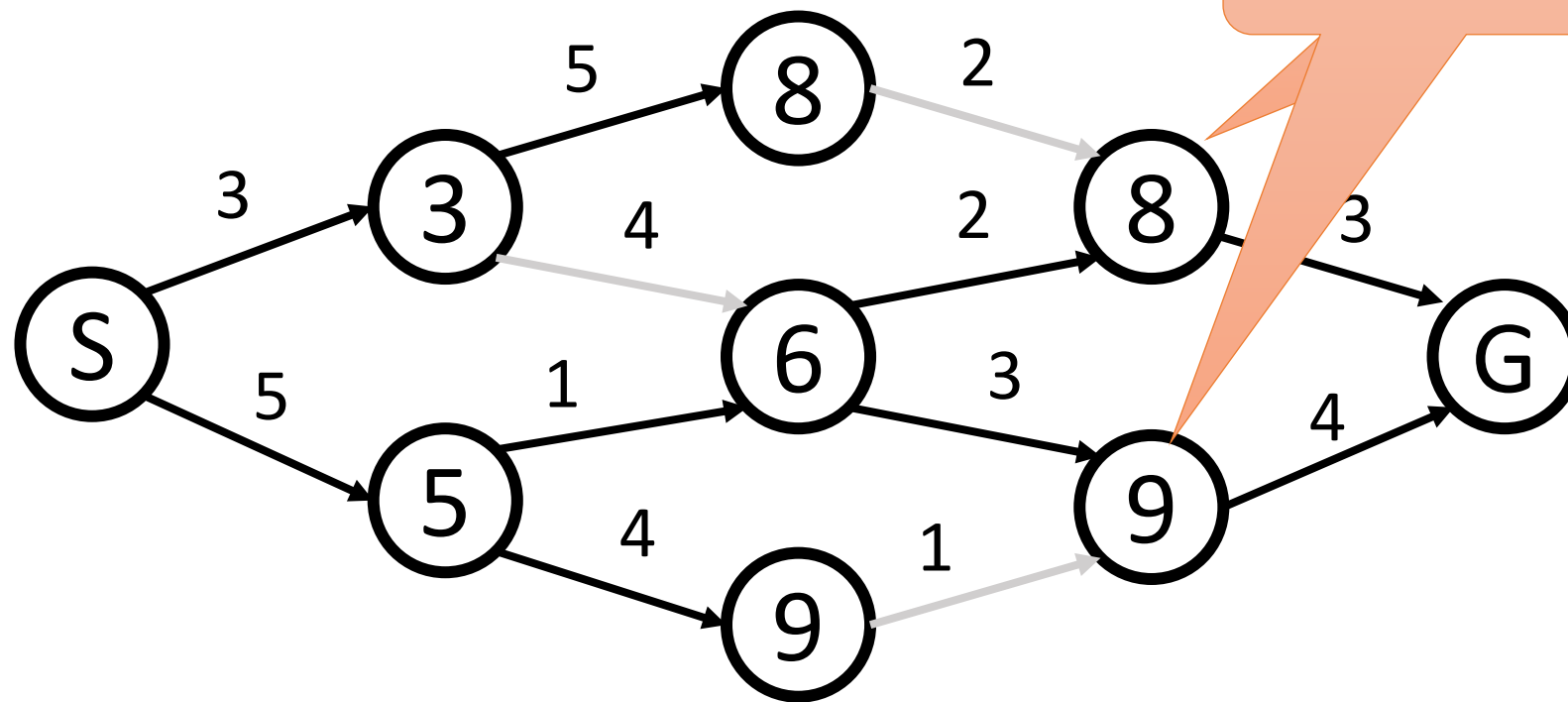
動的計画法の復習

- 数字は移動コスト
- SからGへのコスト最小のルートを探す



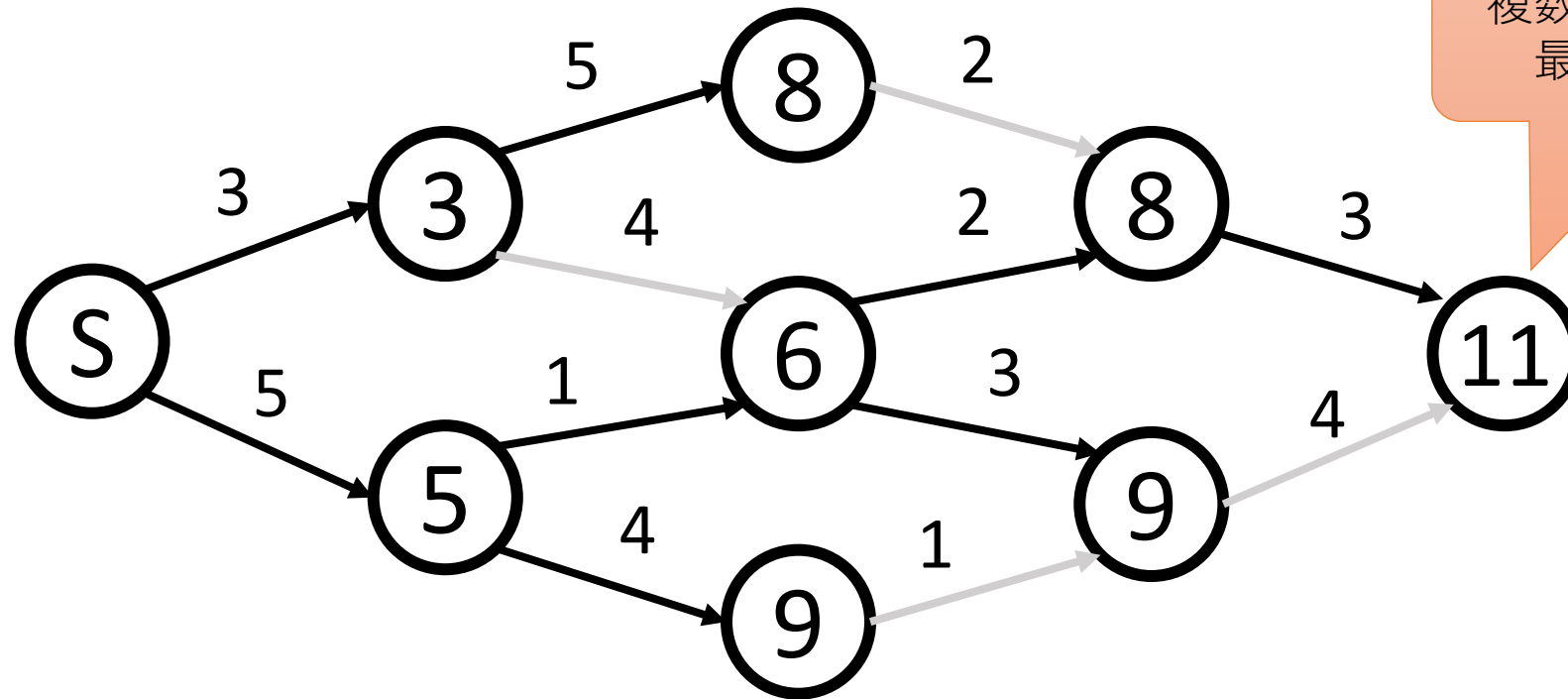
動的計画法の復習

- 数字は移動コスト
- SからGへのコスト最小のルートを探す



動的計画法の復習

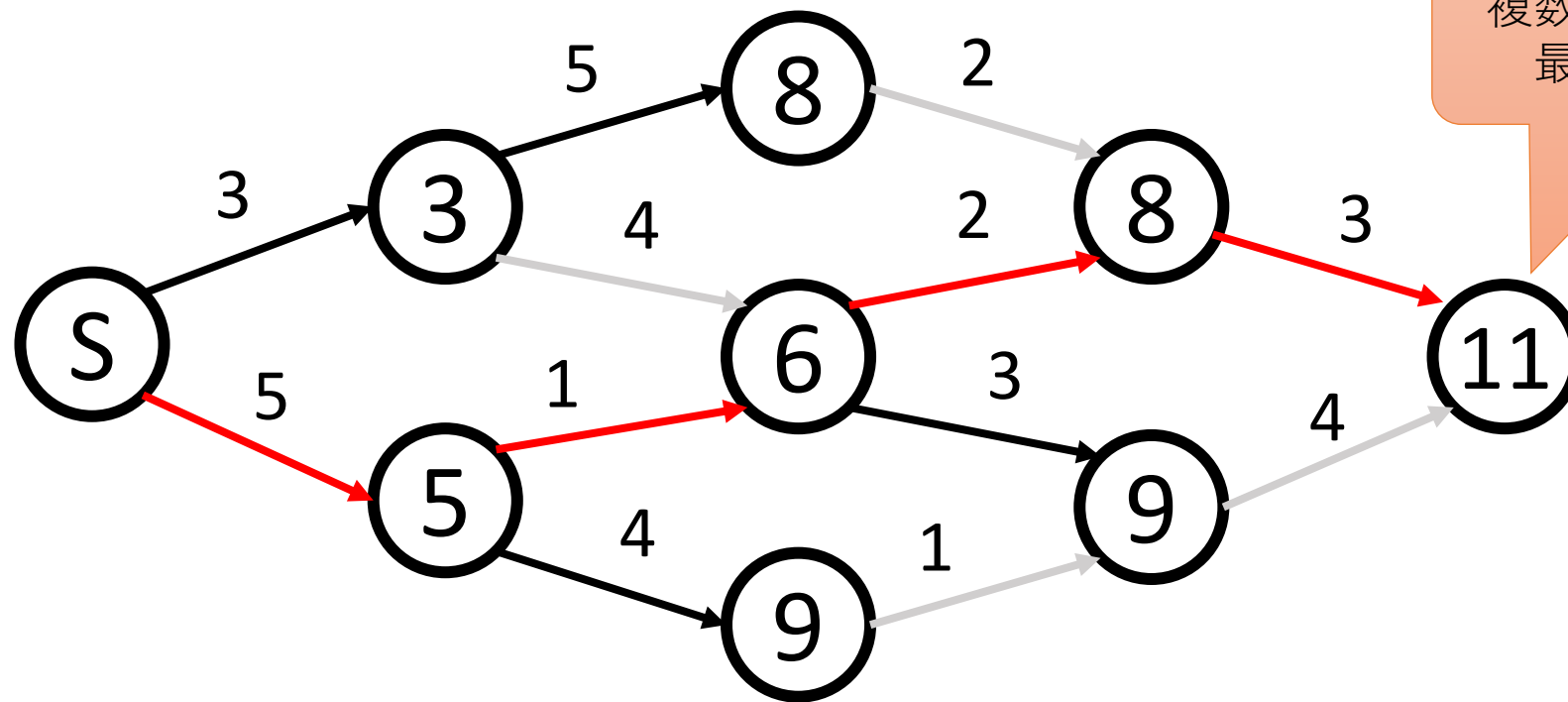
- 数字は移動コスト
- SからGへのコスト最小のルートを探す



複数ルートある場合は
最小コストを優先

動的計画法の復習

- 数字は移動コスト
- SからGへのコスト最小のルートを探す



複数ルートある場合は
最小コストを優先

Q学習と動的計画法の違い

- 動的計画法はすべての状態と行動を網羅して最適解を求める
 - 普通は学習途中の状態を利用することはない
- Q学習は実際に行動してみても学習する
 - すべての状態にたどり着くとは限らない
 - 学習しながら利用することが前提
 - 実際にやってみないとわからない問題にも対応可能

Q学習の実装

- Qテーブル
 - 状態と行動の組み合わせの「評価値」
 - ゴールまでの合計報酬（のようなもの）
- 行動すると、次の状態と報酬がわかる
 - 行動しないで報酬をのぞき見るのはナシだが、Qテーブルは見て良い
 - 次の状態とすべての行動のQテーブルを参照し評価値の最大を見る
 - 「今回の報酬 + 次以降の評価値の最大」を「今回の行動」の評価値としてQテーブルを更新
- これを繰り返すことで、徐々にQテーブルが最適解に近付く
- 基本的には評価値の高い行動を行うが、稀にランダムに行動することで未知の状態を「探索」する

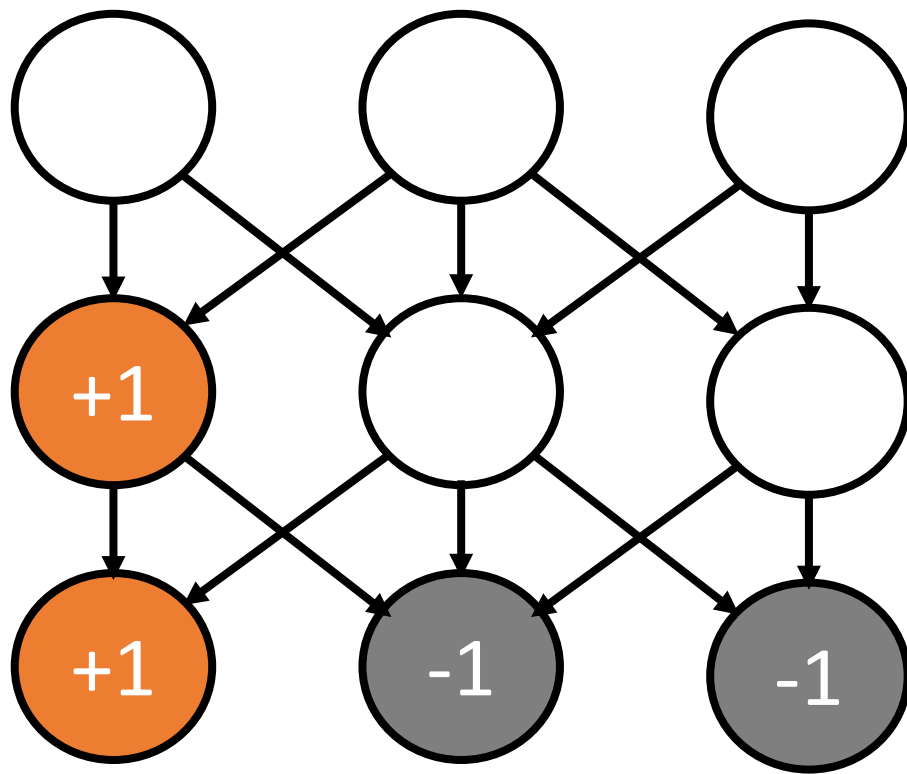
最適解の導出

- 現在の状態からのすべての行動のQテーブルの値を参照し
評価値が最大の行動を選ぶ
 - をゴールまで繰り返すだけ

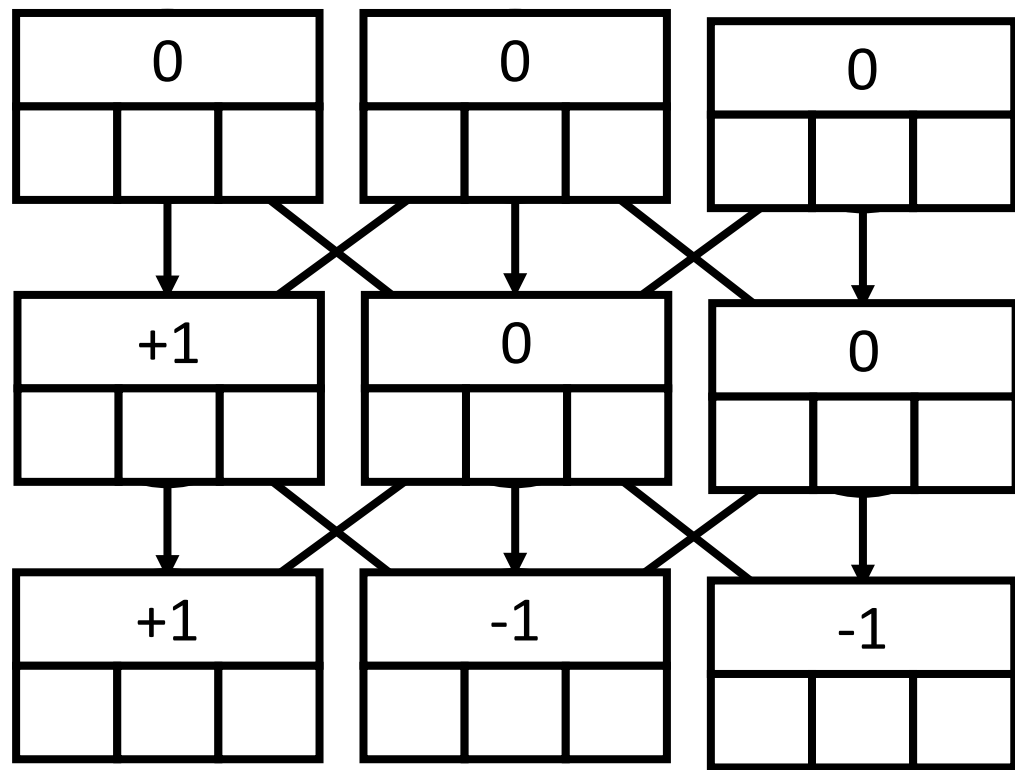
Q学習の例

報酬		
左移動 のQ値	直進 のQ値	右移動 のQ値

Q学習の例



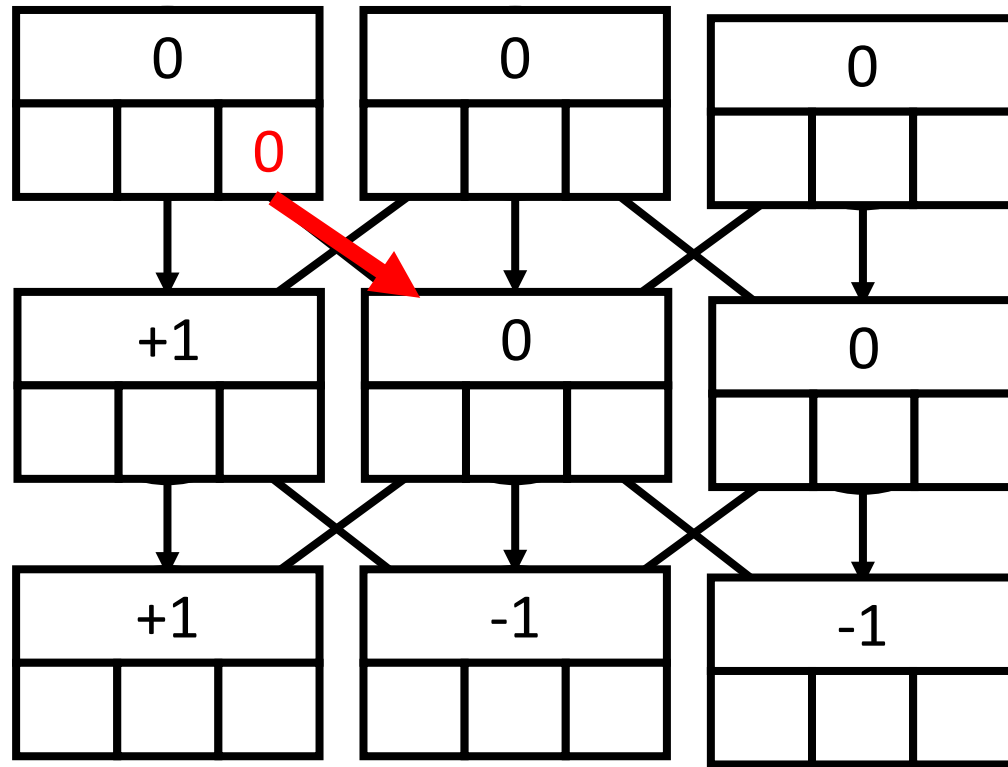
Q学習の例



Q学習の例（学習率=0.5、割引率=0.8）

$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \max_{\text{next_q_max}}) * \alpha$$

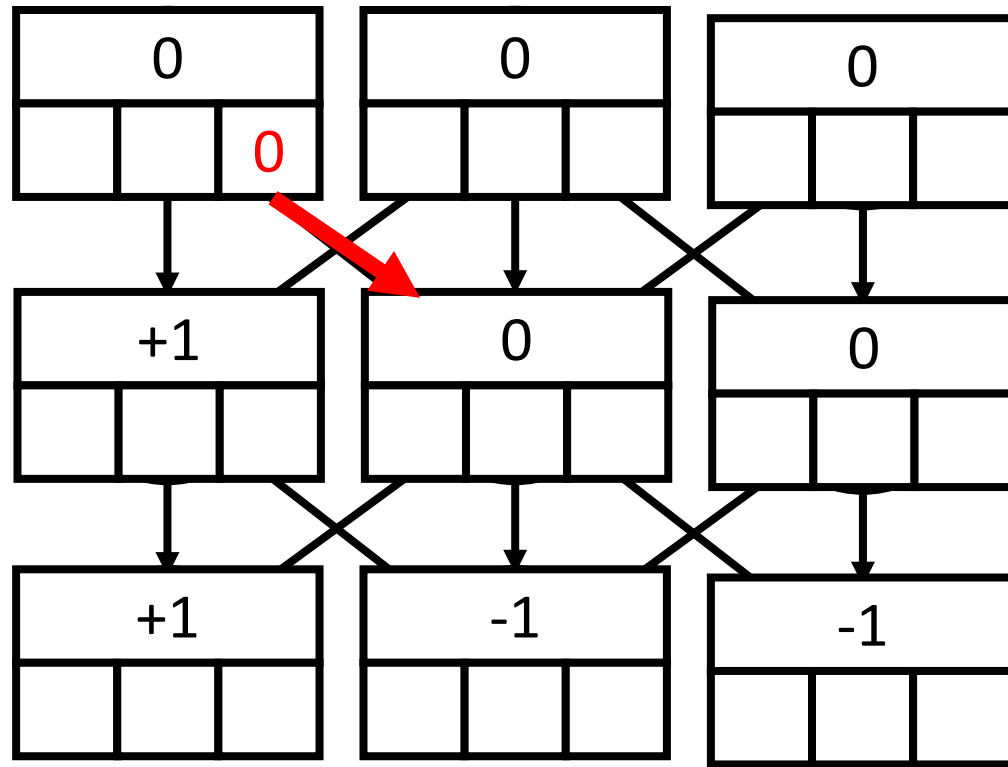
$$\text{Q値} = \text{Q値} * (1 - \text{学習率}) + (\text{報酬} + \text{割引率} * \text{次のQ値の最大}) * (\text{学習率})$$



Q学習の例（学習率=0.5、割引率=0.8）

$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \max_{\text{next_q_max}}) * \alpha$$

$$\begin{aligned} \text{Q値} &= \text{Q値} * (0.5) + \\ &(\text{報酬} + 0.8 * \text{次のQ値の最大}) \\ &* (0.5) \end{aligned}$$

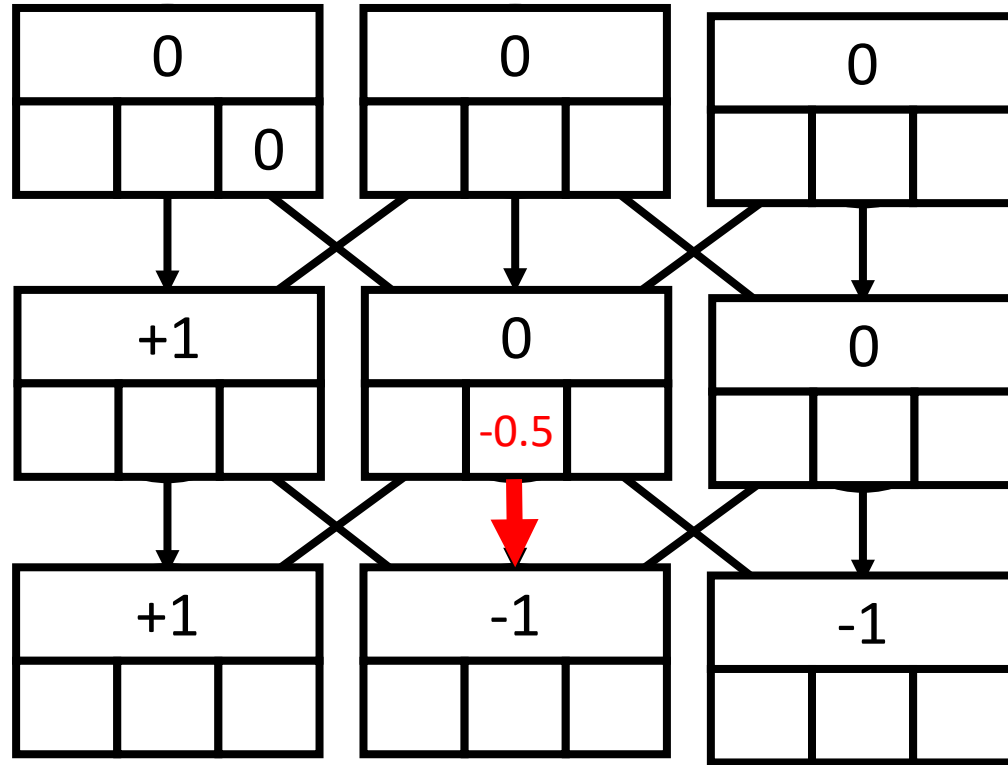


Q学習の例（学習率=0.5、割引率=0.8）

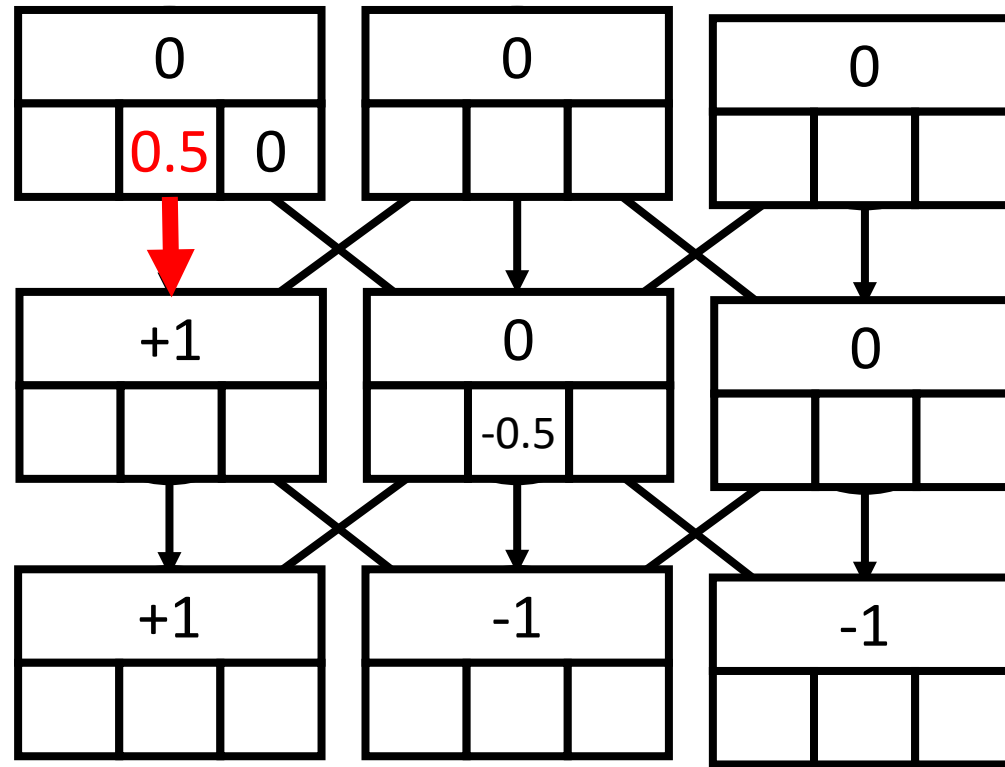
$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \max_{\text{next_q_max}}) * \alpha$$

$$\begin{aligned} \text{Q値} &= \text{Q値} * (0.5) + \\ &(\text{報酬} + 0.8 * \text{次のQ値の最大}) \\ &\quad * (0.5) \end{aligned}$$

$$\begin{aligned} \text{Q値} &= 0 * (0.5) + \\ &(-1 + 0.8 * 0) \\ &\quad * (0.5) \\ &= -0.5 \end{aligned}$$



Q学習の例（学習率=0.5、割引率=0.8）

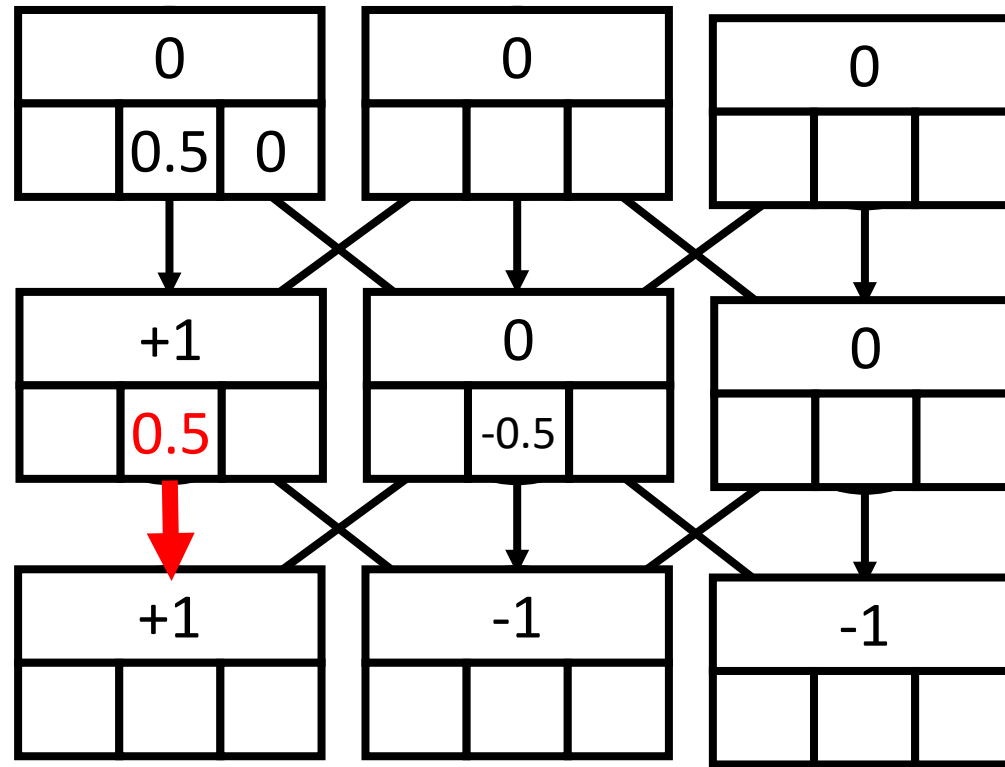


$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \text{next_q_max}) * \alpha$$

$$\begin{aligned} \text{Q値} &= \text{Q値} * (0.5) + \\ &(\text{報酬} + 0.8 * \text{次のQ値の最大}) \\ &\quad * (0.5) \end{aligned}$$

$$\begin{aligned} \text{Q値} &= 0 * (0.5) + \\ &(1 + 0.8 * 0) \\ &\quad * (0.5) \\ &= 0.5 \end{aligned}$$

Q学習の例（学習率=0.5、割引率=0.8）



$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \text{next_q_max}) * \alpha$$

$$\begin{aligned} \text{Q値} &= \text{Q値} * (0.5) + \\ &(\text{報酬} + 0.8 * \text{次のQ値の最大}) \\ &\quad * (0.5) \end{aligned}$$

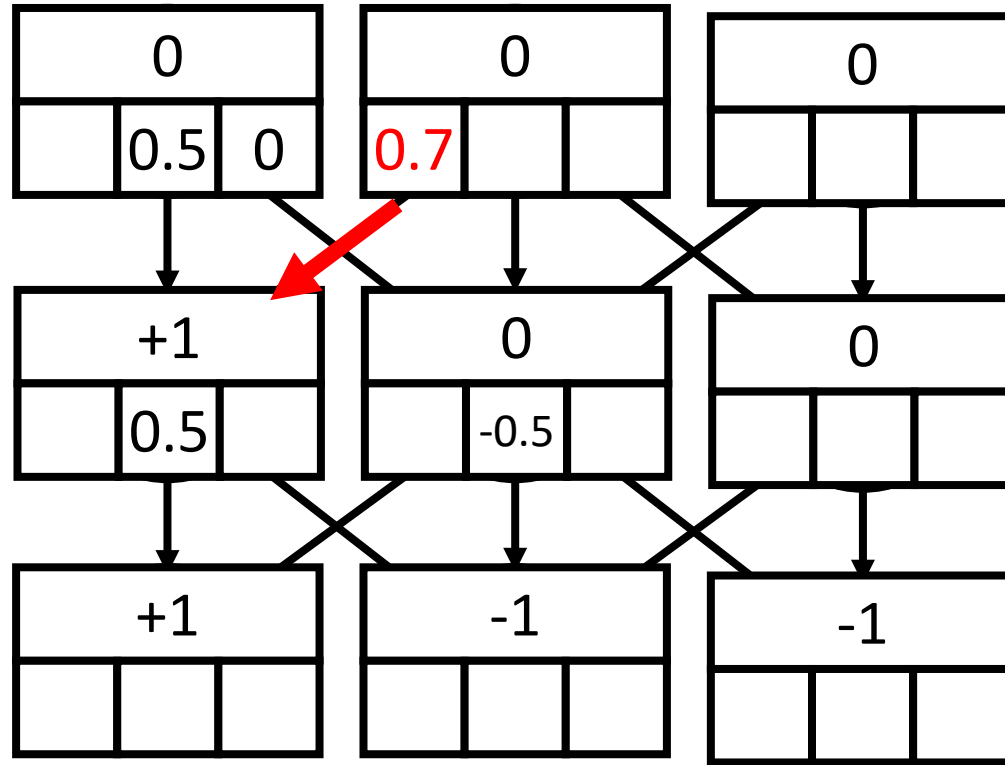
$$\begin{aligned} \text{Q値} &= 0 * (0.5) + \\ &(1 + 0.8 * 0) \\ &\quad * (0.5) \\ &= 0.5 \end{aligned}$$

Q学習の例（学習率=0.5、割引率=0.8）

$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \text{next_q_max}) * \alpha$$

$$\begin{aligned} \text{Q値} &= \text{Q値} * (0.5) + \\ &(\text{報酬} + 0.8 * \text{次のQ値の最大}) \\ &\quad * (0.5) \end{aligned}$$

$$\begin{aligned} \text{Q値} &= 0 * (0.5) + \\ &(1 + 0.8 * 0.5) \\ &\quad * (0.5) \\ &= 0.7 \end{aligned}$$

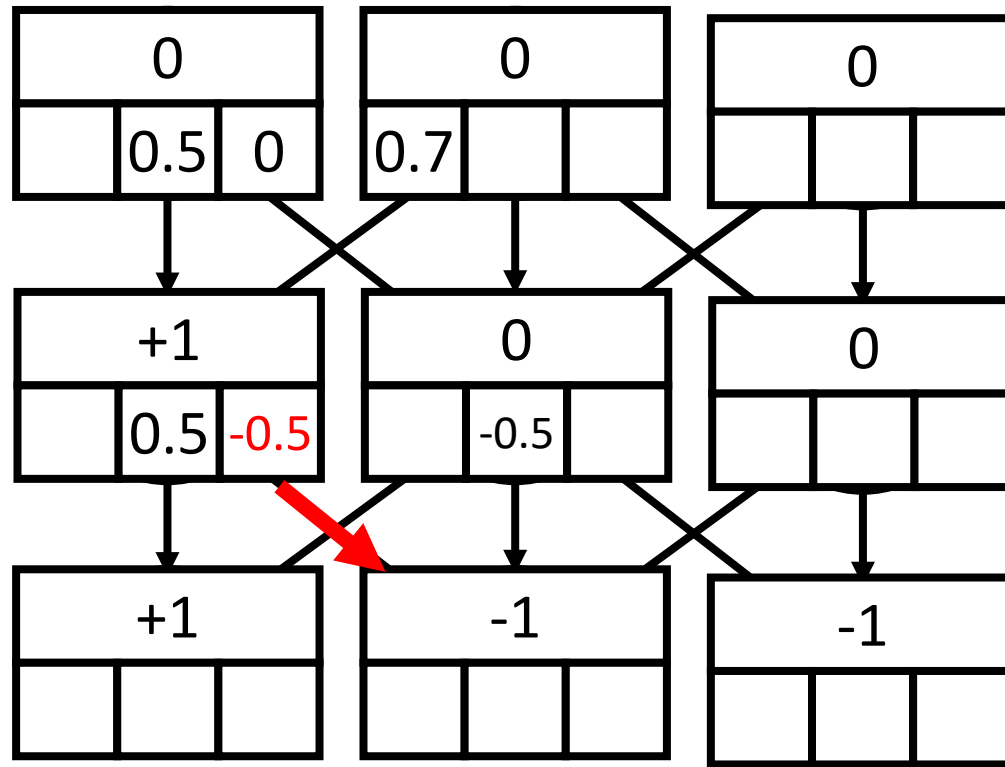


Q学習の例（学習率=0.5、割引率=0.8）

$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \max_{a'} Q[\text{next_state}, a']) * \alpha$$

$$\begin{aligned} \text{Q値} &= \text{Q値} * (0.5) + \\ &(\text{報酬} + 0.8 * \text{次のQ値の最大}) \\ &\quad * (0.5) \end{aligned}$$

$$\begin{aligned} \text{Q値} &= 0 * (0.5) + \\ &(-1 + 0 * 0.5) \\ &\quad * (0.5) \\ &= -0.5 \end{aligned}$$

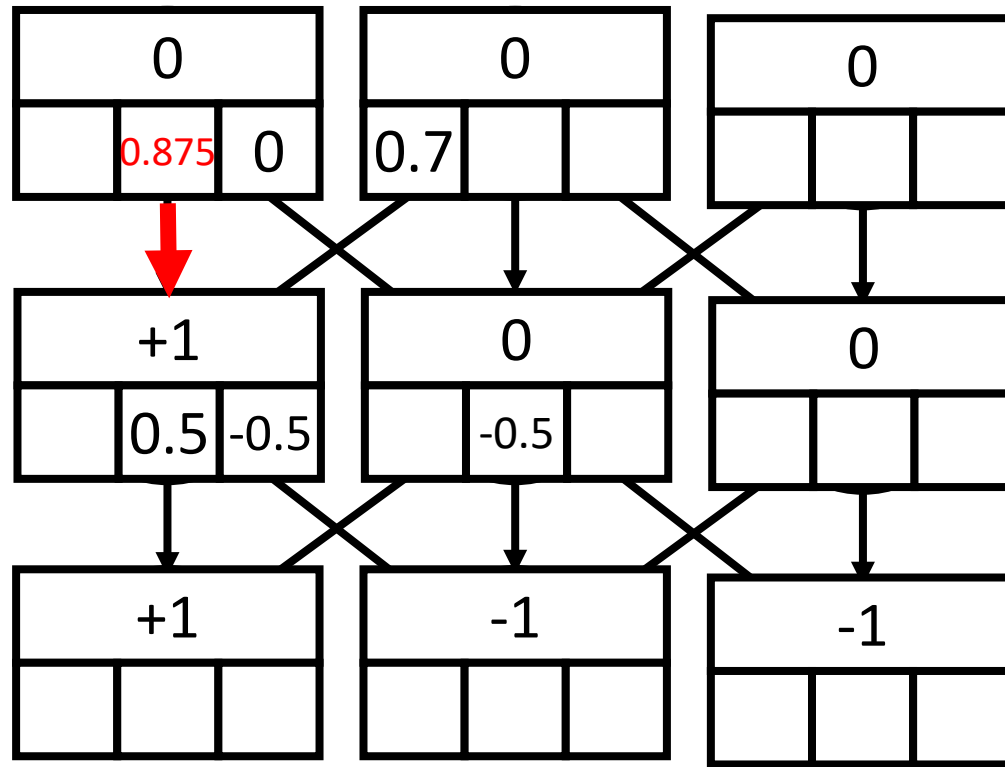


Q学習の例（学習率=0.5、割引率=0.8）

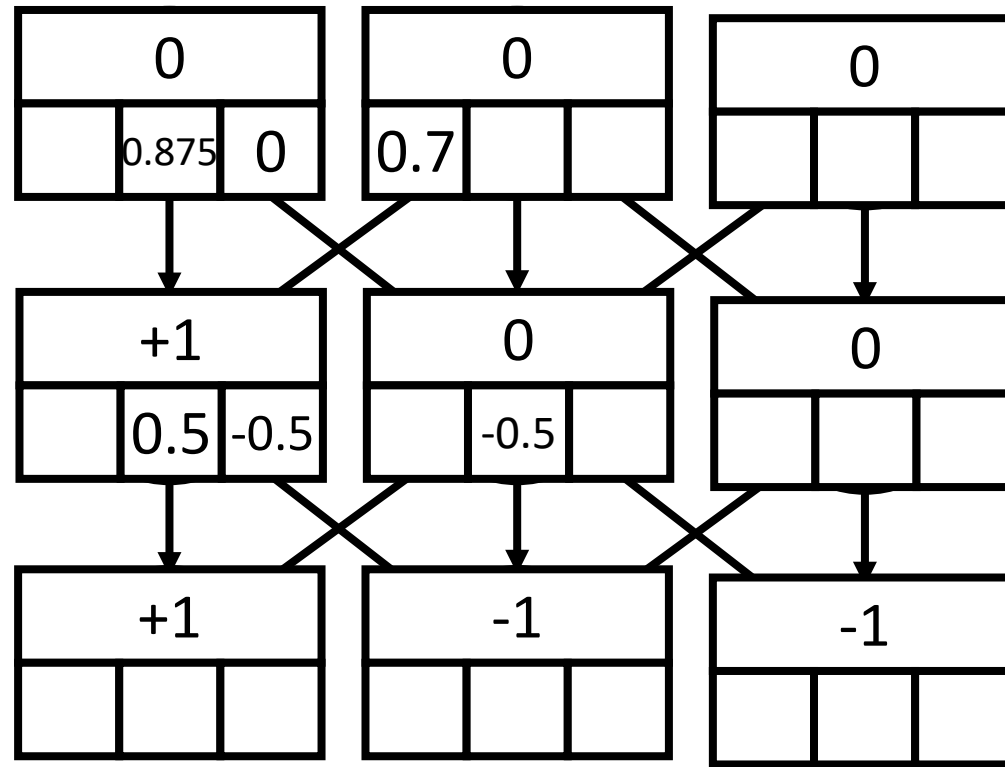
$$Q[\text{state}, \text{action}] = Q[\text{state}, \text{action}] * (1 - \alpha) + (\text{reward} + \gamma * \max_{\text{next_q_max}}) * \alpha$$

$$\begin{aligned} \text{Q値} &= \text{Q値} * (0.5) + \\ &(\text{報酬} + 0.8 * \text{次のQ値の最大}) \\ &* (0.5) \end{aligned}$$

$$\begin{aligned} \text{Q値} &= 0.5 * (0.5) + \\ &(1 + 0.5 * 0.5) \\ &* (0.5) \\ &= 0.875 \end{aligned}$$



Q学習の例（学習率=0.5、割引率=0.8）



報酬1に向かう行動のQ値は大きくなる
報酬-1に向かう行動のQ値は小さくなる

大きいQ値に向かう行動のQ値は大きくなる
小さいQ値に向かう行動のQ値は小さくなる

十分学習を繰り返すと、、、
大きいQ値の行動を選択すると
1にたどり着くようになる

Q学習と不完全情報ゲーム

- 不完全情報ゲーム
 - 一部の情報がプレイヤーから観測不可能
 - 多くのトランプゲームやボードゲームなど
 - ↔ 囲碁や将棋は完全情報ゲーム
- 観測可能な「状態」が同じでも、観測不可能な状態が異なる可能性があるので、最適な行動が一意に定まらない
 - 学習してもQテーブルが収束しない
 - ナッシュ均衡
 - プレイヤー全員があらゆる可能性を考えて最適な行動を行うならば、その行動は一意に定まる
 - 最適な行動をしないプレイヤーがいると最適な行動が定まらない

Q学習の演習

- 「環境」の作成が一番大変
 - 未知の問題を解きたい場合は、自分で環境を作成することが必須
- 学習アルゴリズムに修正の余地はあまりない
 - まだDQN（Deep Q-Network）ではないため
 - 学習率とか割引率とかあるけど、これらの最適化は本質的ではない
- 実質的に環境構築の演習
を通してQ学習を理解する
- Q-learning2.py を参考にする

実行例

- 最短ルートで報酬を獲得している
 - x 壁：当たると即終了
 - + 報酬：+1
 - o 現在地
 - * 現在地 + 報酬
 - X 現在地 + 壁
 - 右端の数字は報酬

スクリーンショットが古く
現在は*ではなくo

55	#	x	x	x	*	#	1.0
56	#	+				#	0.0
57	#	+				#	0.0
58	#	x				#	1.0
59	#	x				#	1.0
60	#	x	x	x		#	1.0
61	#				+	#	0.0
62	#				+	#	0.0
63	#				*	#	1.0
64	#				*	#	1.0
65	#	x	x	x	*	#	1.0
66	#			*		#	1.0
67	#					#	1.0

☆課題5 (体裁で+2点)

- 共通：目標は新ルールで学習して200ステップゴール
- レベル-1 (3点)
 - 左右の壁をなくして、落ちたら即ゲームオーバーにする
- レベル0 (4点)
 - コースの幅を7以上にする
- レベル1 (5点)
 - 左の壁を戻して、画面を左に90度回転し、地面とし、重力を作る
 - 「地面」には穴を開けておく
 - 移動の代わりに「ジャンプ」を実装する
- レベル2 (6点)
 - 空中ジャンプと土管を実装してflappy birdにする (GUIは不要)

<https://flappybird.io/> を参照

レベル1の実行例

- 穴と壁を回避している
 - ジャンプ1回で高さ2、幅4のジャンプをしている
 - 高さ1の空中ジャンプを2回しているわけではない
 - ジャンプを始めたらず地まで操作できない
- もっとカッコイイ表示を期待

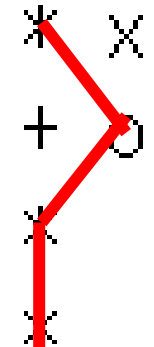
補足

- 「運良く」クリアすることがあるが、ちゃんと正しく動作していることを見極めること
 - 対応出来ないパターンが出てこなくなるまで実行を繰り返す等
 - 運だけではないことがわかるレポートにしてください
- 例えば壁を取り除いたり、「右端」が運良く出なかったときの結果だけを掲載する等
- 幅を広げたが、一度も広げた部分に行かない
 - 逆に広げた部分には壁が無いので、一度移動したら動かない等

任意課題2（合計5点）

- サンプルのルールでは「穴」の列の次の移動が最適化されていないように見える
 - 右の例では直進すれば良い状況なのに+を避けている（ように見える）
- 原因を説明せよ（1点）
- 改良案を説明せよ（2点）
 - ソースコードのどこを変えてもよい
- 改良案を実装し避けないことを実証せよ（2点）

10	#	x	x	*	x	#
11	#			+	o	#
12	#			*		#
13	#			*		#



締切

2025/6/12 23:59

(予告) ☆課題6 (体裁で+2点)

- 共通：目標は新ルールで学習して100ステップゴール
- マップ中に**ランダムで障害物を作り**、移動中にぶつかるとゲームオーバーとする
 - ランダムは運が悪いとクリアできないパターンができるので注意
- **DQNで実装**