ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ВШ программной инженерии

ПОЛИТЕХ
Санкт-Петербургский
политехнический университет
Петра Великого

КУРСОВАЯ РАБОТА

Использование искусственного интеллекта для прохождения игры "Змейка"

по дисциплине «Глубокое обучение»

Выполнили
Студенты 3530202/80202 группы — А.М. Потапова, Р. Ж. Айдаров, Н.С. Парфенов

Руководитель — О.Г. Малеев

Санкт-Петербург
2023 г.

Содержание

Введение

Змейка[1] — компьютерная игра, уходящая корнями в системы 1970-х. Наиболее известна версия от Nokia, впервые появившаяся в кнопочном телефоне Nokia 6110. Разработана финляндским разработчиком Танели Арманто.

Игрок управляет длинным, тонким существом, напоминающим змею, которое ползает по плоскости (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля (существуют варианты, где при прохождении через край змея выходит из противоположного края поля). Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. В другом варианте двое играют двумя такими змеями так, чтобы вынудить соперника врезаться во что-то. В нашей работы мы реализуем классическую версию.

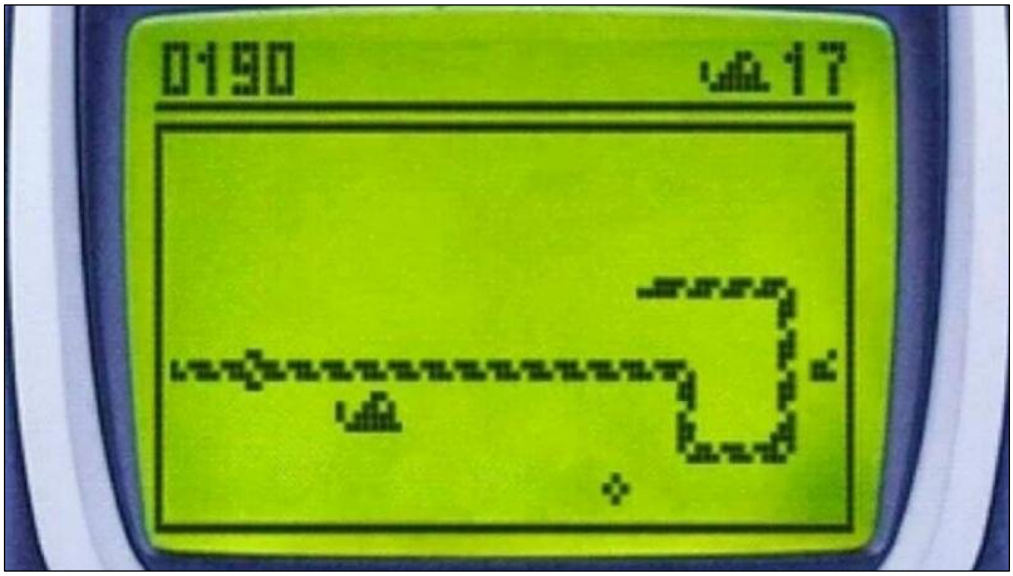

Рисунок 1. Интерфейс игры змейка на Nokia 6110.

Структура нейронной сети

Нейронная сеть – многослойный персептрон.

Входной слой – 11 нейронов.

Скрытый слов – число нейронов варьируется.

Выходной – 4 нейрона.

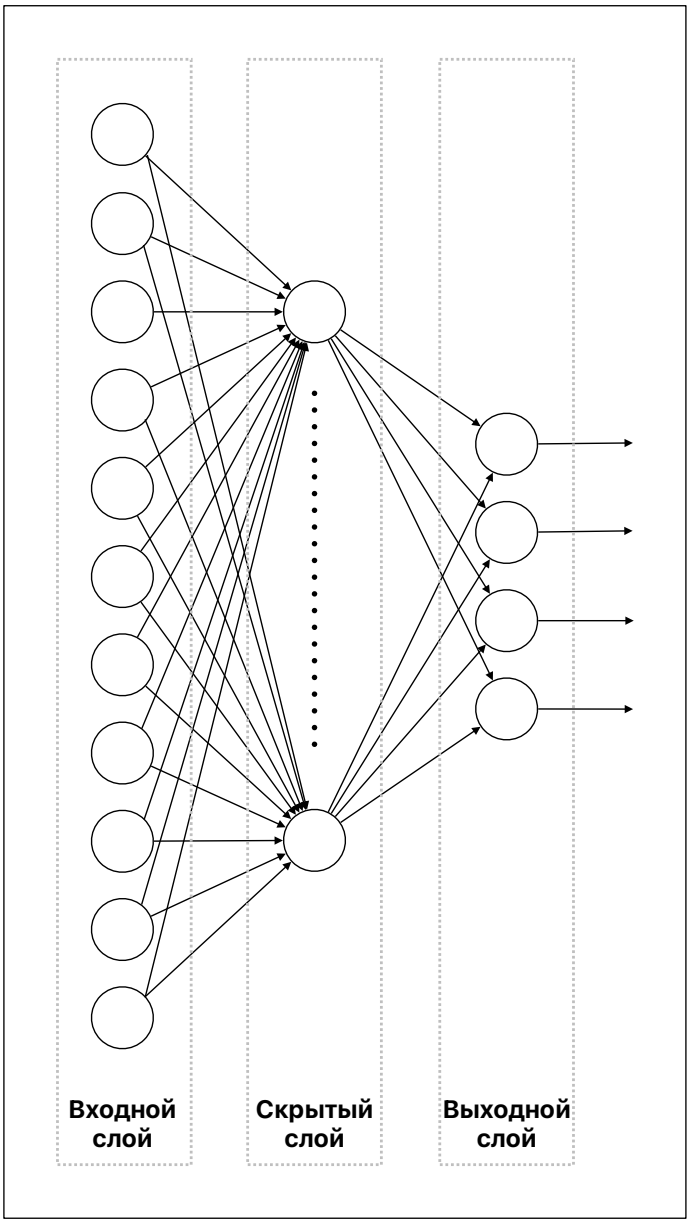

Рисунок 2. Структура многослойного персептрона.

Реализация

Для реализации был выбран фреймворк PyTorch[2] и язык программирования Python[3]. PyTorch - это библиотека машинного обучения с открытым исходным кодом, которая предоставляет инструменты для создания и обучения глубоких нейронных сетей. Она имеет несколько преимуществ перед другими библиотеками машинного обучения, такими как TensorFlow[4]:

1. Простота и интуитивность: PyTorch был разработан для облегчения разработки нейронных сетей. Он имеет простой и интуитивный интерфейс, который позволяет пользователям быстро создавать и обучать модели машинного обучения.
2. Динамический граф вычислений: PyTorch использует динамический граф вычислений, что делает его более гибким и удобным для работы. Это позволяет пользователям изменять граф вычислений на лету, что облегчает отладку и эксперименты.
3. Простота отладки: PyTorch предоставляет простой и удобный интерфейс для отладки моделей машинного обучения. Это позволяет пользователям быстро находить ошибки в своих моделях и исправлять их.
4. Компактный код: PyTorch позволяет писать компактный и выразительный код. Это делает код более читабельным и понятным, что упрощает совместную работу и обмен кодом с другими разработчиками.
5. Наличие предобученных моделей: PyTorch предоставляет доступ к большому количеству предобученных моделей машинного обучения, что упрощает создание и обучение собственных моделей.
6. Активное сообщество: PyTorch имеет большое и активное сообщество разработчиков, которые постоянно работают над улучшением библиотеки и предоставляют полезные ресурсы и советы для пользователей.

Каждое состояние имеет вид тензора из 11 элементов, где первые три говорят о ближайшей опасности, в остальных мы сообщаем информацию о текущем направлении змейки и о том, где, относительно нее находится яблоко (выше, ниже, левее или правее)

Класс, описывающий многослойный персептрон представлен на рисунке ниже:

```
6   class Linear_QNet(nn.Module):
7       def __init__(self, input_size: int, hidden_size: int, output_size: int):
8           super().__init__()
9           self.linear1 = nn.Linear(input_size, hidden_size)
10          self.linear2 = nn.Linear(hidden_size, output_size)
11
12      def forward(self, x):
13          return self.linear2(nn.functional.relu(self.linear1(x)))
```

Рисунок 3. Класс описывающий многослойный персептрон.

Качество предсказания многослойного персептрона зависит от разных факторов, включая количество нейронов в скрытом слое. Когда количество нейронов в скрытом слое увеличивается, у модели появляется больше свободы для выучивания более сложных функций, что может улучшить качество предсказания.

Однако, если количество нейронов в скрытом слое слишком велико, то модель может стать переобученной, то есть она будет слишком хорошо запоминать обучающие данные, что может привести к плохому качеству предсказания на новых данных. С другой стороны, если количество нейронов в скрытом слое слишком мало, то модель может недообучаться, то есть она будет недостаточно хорошо аппроксимировать зависимости в данных, что также приведет к плохому качеству предсказания.

Соответственно, чтобы получить оптимальное качество предсказания, необходимо подобрать оптимальное количество нейронов в скрытом слое.

Обучение

Обучение с подкреплением q сети[4] – метод, применяемый в искусственном интеллекте при агентном подходе. Относится к экспериментам вида обучение с подкреплением. На основе получаемого от среды вознаграждения агент формирует функцию полезности Q, что впоследствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой. Одно из преимуществ Q-обучения — то, что оно в состоянии сравнить ожидаемую полезность доступных действий, не формируя модели окружающей среды. Применяется для ситуаций, которые можно представить в виде марковского процесса принятия решений.

Агент и Среда играют ключевые роли в алгоритме обучения с подкреплением. Среда – это тот мир, в котором приходится выживать Агенту. Кроме того, Агент получает от Среды подкрепляющие сигналы (вознаграждение): это число, характеризующее, насколько хорошим или плохим можно считать текущее состояние мира. Цель Агента — максимизировать совокупное вознаграждение, так называемый «выигрыш».

1. **Состояния**: Состояние – это полное описание мира, в котором не упущено ни единого фрагмента информации, характеризующей этот мир. Это может быть позиция, фиксированная или динамическая. Как правило, такие состояния записываются в виде массивов, матриц или тензоров высшего порядка.
2. **Действие**: Действие обычно зависит от условий окружающей среды, и в различных средах агент будет предпринимать разные действия. Множество допустимых действий агента записывается в пространстве, именуемом «пространство действий». Как правило, количество действий в пространстве конечно.
3. **Среда**: Место, в котором агент существует и с которым взаимодействует. Для различных сред используются различные типы вознаграждений, стратегий и т.д.

4. **Вознаграждение** и **выигрыш**: Отслеживание функции вознаграждения R при обучении с подкреплением происходит постоянно. Она критически важна при настройке алгоритма, его оптимизации, а также при прекращении обучения. Она зависит от текущего состояния мира, только что предпринятого действия и следующего состояния мира.

5. **Стратегии**: стратегия — это правило, в соответствии с которым агент избирает следующее действие. Набор стратегий также именуется «мозгом» агента.

Класс для Q обучения нейросети реализованной в нашем проекте представлен на рисунке ниже.

```python
class QTrainer:
    def __init__(self, model, lr, gamma):
        self.lr = lr
        self.gamma = gamma
        self.model = model
        self.optimizer = optim.Adam(model.parameters(), lr=self.lr)
        self.criterion = nn.MSELoss()

    def train_step(self, state, action, reward, next_state, done):
        state = torch.tensor(state, dtype=torch.float)
        next_state = torch.tensor(next_state, dtype=torch.float)
        action = torch.tensor(action, dtype=torch.long)
        reward = torch.tensor(reward, dtype=torch.float)

        if len(state.shape) == 1:
            state = torch.unsqueeze(state, 0)
            next_state = torch.unsqueeze(next_state, 0)
            action = torch.unsqueeze(action, 0)
            reward = torch.unsqueeze(reward, 0)
            done = (done, )

        pred = self.model(state)

        target = pred.clone()
        for idx in range(len(done)):
            Q_new = reward[idx]
            if not done[idx]:
                Q_new = reward[idx] + self.gamma * \
                    torch.max(self.model(next_state[idx]))

            target[idx][torch.argmax(action[idx]).item()] = Q_new

        self.optimizer.zero_grad()
        loss = self.criterion(target, pred)
        loss.backward()

        self.optimizer.step()
```

Рисунок 4. Класс QTrainer.

Результат

В ходе курсовой работы была разработана игра «Змейка». Интерфейс представлен на рисунке ниже.

Рисунок 5. Интерфейс игры «Змейка»

Таблица 1 – Зависимость результата от количества нейроном в скрытом слое

Количество нейронов в скрытом слое	Рекорд, поставленный нейронной сетью
32	2
64	3
128	56
256	69
512	88
1024	102 и более

Как можно заметить, результаты со значениями 32 и 64 нельзя назвать оптимальными. При больших значениях агент показывает достаточно неплохой результат. Так же, стоит отметить, что при увеличении количества нейронов, рекордное значение возрастает. Таким образом, увеличивая количество нейронов, можно поставить абсолютный рекорд.

Список литературы

1. Snake (игра) – Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Snake_(игра)
2. PyTorch Documentation – PyTorch.org [Электронный ресурс]. URL: https://pytorch.org/docs/stable/index.html
3. Python Documentation – Python.org [Электронный ресурс]. URL: https://docs.python.org/3/
4. TenserFlow – TenserFlow.org [Электронный ресурс]. URL: https://www.tensorflow.org/?hl=ru
5. Обучение с подкреплением на языке Python – Habr.com [Электронный ресурс]. URL: https://habr.com/ru/company/piter/blog/434738/