Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Курсов Проект

на тема: „**TextWorld Problem: A Reinforcement and Language Learning Challenge**”

Студент: **Николай Димитров Бабулков** Ф.Н. **26047**
Курс: „Изкуствен интелект“, Учебна година: 2018/19

Студент: **Мартин Георгиев Георгиев** Ф.Н. **24907**

Курс: „Изкуствен интелект“, Учебна година: 2018/19

Съдържание

1. Резюме

Текстовите игри могат да изглеждат примитивни, сравнени с графичните игри в днешно време, но за да прогресират дори в най-простата такава игра, хората трябва да използват специален набор от умения, като безпроблемно разбираме езикови описания, планираме следващите си действия, изследване на околната среда, запомняне важна информация и обобщаване миналия опит. Агентите с изкуствен интелект все още не притежават напълно такива способности, но скорошни разработки в сферата на машинното самообучение позволяват да се достигнат невиждани преди резулатати върху проблеми от този тип. Нашият подход към проблема е да използваме последните нововъведения в сферата на разбирането на естествен език, заедно с утвърдени решения от обучението с подсилване(Sutton and Barto, 2018), за да покажем, че TextWorld проблемът (Côté et al., 2018) е ефективно решим.

2. Въведение

Формално дефинирано, играенето на текстова игра от агент представлява разбиране на текст с цел, предприемане на действия в среда, така че да се максимизира някакво кумулативно възнаграждение за определен брой ходове. Конкретно за състезанието TextWorld (Côté et al., 2018), основната идея е да се създаде агент, който може да играе ефективно и да печели опростени текстови игри, представляващи сбор от цели, за всяка от които агентът получава награди.

Целите са сготвяне на разнообразни ястия с налични в средата продукти. Агентът получава на входа си цели, които трябва да изпълни, заедно с текстово описание на текуща ситуация, в която се намира. След всяко свое действие, той получава описание на резултата от действието му и награда за него. Наградата може да бъде 1, ако действието му го доближава до изпълнение на целите, или 0 - в обратния случай. Всяка информация, подадена на агента, освен наградата, е описание на естествен език, точно както при човек, играещ играта.

Действията, изпълнявани в средата, са текстови команди, които в оптималния случаи ще доведе до получаване на позитивна награда. Пример за изход на агента могат да са: “open glass door”, “take red pepper from table”, “chop carrot with knife”, “prepare meal” и т.н.

Структурата на всяка команда представлява:

глагол [прилагателни_1] съществително_1 [предлог] [прилагателни_2] [съществително_2]

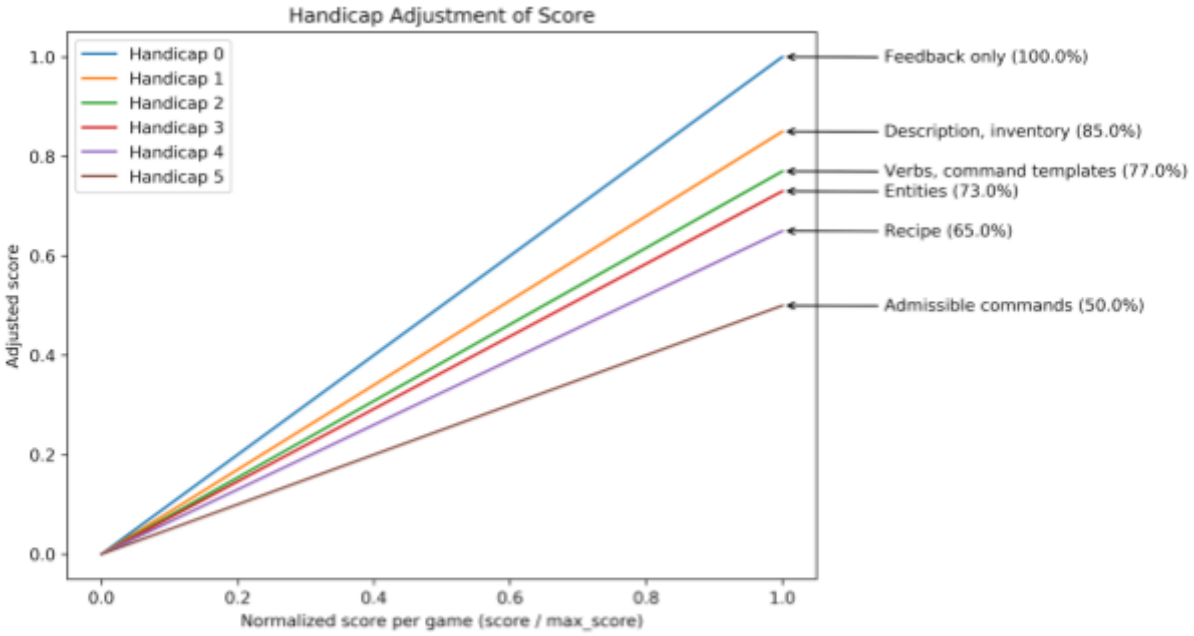
Описание	**Примерен отговор от средата/Вход за агента**
Текущо опсание на средата	Look at you, bigshot, walking into a kitchen like it isn't some huge deal. You make out a fridge. I mean, just wow! Isn't TextWorld just the best? You see a closed oven. You can make out a table...
Текуща цел	You are hungry! Let's cook a delicious meal. Check the cookbook in the kitchen for the recipe. Once done, enjoy your meal!
Резултат от предишното действие на агента	You open the fridge, revealing a fried yellow bell pepper, an orange bell pepper and a cilantro.
Текущ инвентар	You are carrying: a knife a fried yellow potato a cookbook

Това улеснява генерирането на действия, тъй като ограничава пространството от възможни действия, от всички възможни смислени изречения, до такива, с описаната горе форма. Така задачата на агента, от генериране на свободен текст, се превръща в класификация на правилната дума на съответната позиция.

Пълната среда, в която агентът трябва да се ориентира, са една или повече отделни локации, между някои, от които може да се преминава. Пример за такива са: kitchen, garden, basement и т.н. Всяка една от тези локации има собствено описание и група предмети намиращи се в тях, като всеки от предметите може да е от помощ за достигането на целите на агента.

Поради трудността на проблема, създателите на състезанието дават възможността агентите да изискват допълнителна информация за състоянието на света от играта, въпреки че това води до наказание в зависимост от вида на използваната информация(Фиг. 1). Категорията на агента се определя от най-високата категория искана информация. При тестване на агента, той влиза в една от шест категории, спрямо информацията, която иска от средата.

- Handicap 0: Само отговор на средата за предишното действие на агента
- Handicap 1: Информация за текущата .
  (Отговори на командите: “look” и “Inventory”)
- Handicap 2: Възможни глаголи и шаблони за команди в текущата игра
- Handicap 3: Срещани обекти в текущата игра
- Handicap 4: Рецепти за ястията, които трябва да се сготвят
  (Намират в обекта “готварска книга”(cookbook), който може да се намери в играта)
- Handicap 5: Възможни команди, от които агентът да може избира на всеки ход



Фиг. 1 Наказания върху точките спрямо исканата информация от агента

Нашето решение е да използваме невронни мрежи за разбиране на текста, генериран от TextWorld средата, и предвиждане на бъдещи награди за възможните действия. Използвайки това, намираме най-доброто предвидено действие за текущото състояние и го изпълняваме в средата. На входът си на невронната мрежа получава конкатенация от цялата информация, предоставена от средата. Изходът на модела е са пет паралелни слоя, всеки от които предвижда една или повече думи, в описаната горе форма на команда. Единствените части на речта, които модела не предвижда са предлозите. Те се добавят в зависимост от глагола и това дали има второ съществително в изречението. Моделът може да избере да пропусне, което и да е от прилагателните и второто съществително, като така може да съкрати командата до минимум от две думи - един глагол и едно съществително. Агентите, които създадохме изискват информация от категории, или Handicap 3 или Handicap 4, което значи че получават информация за текущата локация и инвентар, заедно с възможните глаголи, съществителни и прилагателни.

3. Преглед на областта

Двете сфери в машинното самообучение, които имат най-голямо влияние върху решението на играта в състезанието TextWorld са Reinforcement Learning (RL) и Natural Language Understanding (NLU).

3.1. Reinforcement Learning

Основната идея на RL е да се научи как да се съпоставят ситуации на действия, за да се максимизира възнаграждение. На учащия агент не се казва кои действия да се предприеме, но вместо това трябва сам да открие кои действия дават най-голяма награда, като ги изпробва. В най-интересните и предизвикателни случаи,

действията могат да засегнат не само непосредствената награда, но и следващите ситуации, а чрез това, всички последвали награди. Търсене, чрез проба и грешка, и отложено възнаграждение - са двете най-важни отличителни черти на RL. (Sutton and Barto, 2018)

Действията може да са дадени предварително или да се откриват в последствие. Оценява се действието, което сме предприели, и агентът се опитва да максимизира оценката. Тази оценка се нарича награда. Награда може да се дава както на ход така и на завършване на целия процес.

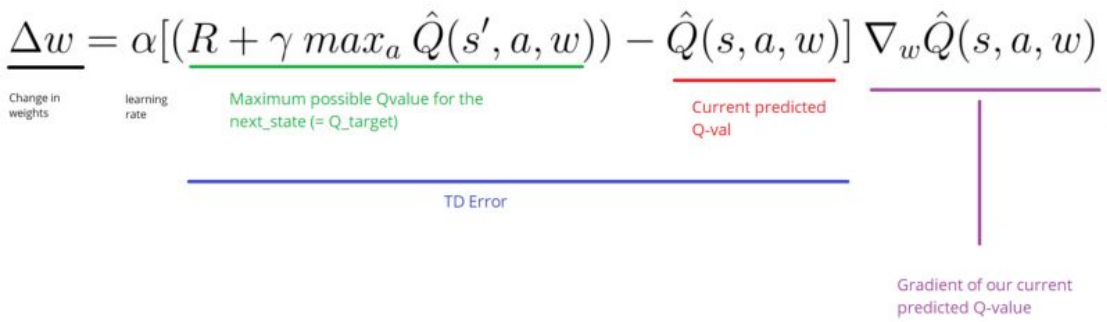
Важна част от RL е изследването на пространството от възможни действия и състояния. Ако този процес не се случи агентът няма да научи за други, по-възнаграждаващи действия от тези, които вече знае. Но ако модела винаги изследва за нови действия, той ще има по-рядък шанс да се възползва от знанията си. Заради това е важно в процеса на обучение да се започва с период на изследване на възможностите, последван от период на експлоатация на наученото. (Sutton and Barto, 2018)

Един от най-успешните практически алгоритми за RL е *Q-learning* (Watkins, 1989). При него се намират стойности *q-values*, които представляват вид “качество” на предприетите действия в текущото състояние. Идеята на алгоритъма е да се научи стратегия за това какви действия да предприема спрямо ситуацията. Целта на стратегията е да се оптимизира наградата, получена от играта за всички стъпки от нея. Средата в *Q-learning*, се формализира като краен процес на Марков за вземане на решения (Markov Decision Process) (Puterman, 1994).

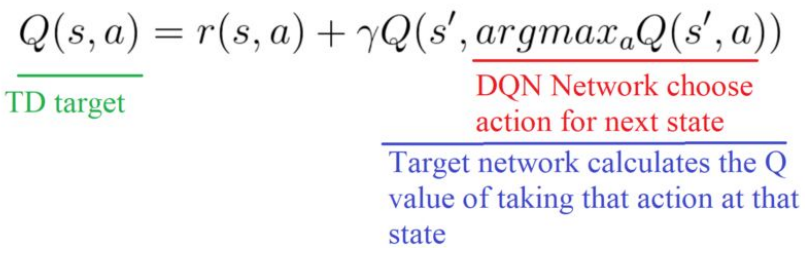
$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

където функцията за изчисляване на q-values е $Q : S \times A \rightarrow \mathbb{R}$

Първоначално Q-learning не е реализирано чрез модели, а основно чрез динамично програмиране. Но с навлизането на модели и по-специфично невронни мрежи, те започват да се използват. За подобряване на поведението на функицята *Q* е създаден алгоритъма *DQN (Deep Q-Network)* (Mnih et al., 2013). При него се използват невронни мрежи за реализирането на *Q-learning* и апроксимацията на *q-values.* Използвайки обратната връзка от околната среда, невронната мрежа може да се учи от разликата между очакваното възнаграждение и реалното възнаграждение. Мрежата коригира своите тегла и така подобрява интерпретацията за това как средата реагира на действията й.



DQN е нестабилен, защото е прекалено оптимистичен в предвижданията си. Това е резултат от един модел, едновременно избиращ и оценяващ своето действие. Едно предложение за предотвратяването на този ефект е *Double Deep Q-Network (DDQN)* (van Hasselt et al., 2016). При този подход се използват две мрежи. Едната служи за намиране на най - подходящо действие в момента, докато другата служи за даване на оценка на избраното действие. Формално обяснение на изчисляването на *q-values* е както следва:



Основен подход, използван за по-бързо обучаване на агенти, е Temporal difference (TD) learning (Sutton, 1988). Методът се изразява в това, че знанията на агента се обновяват на всяка негова стъпка, вместо в края на всеки епизод от играта.

Проблеми с обучението на модели само с текущите им действия е, че моделът ще научи предимно най-често срещаните действия и ще забрави рядко срещани действия, които могат да имат влияние в бъдеще. Решението е повторение на случилото се - Experience Replay (Lin, 1992). Имплементацията му в реални модели се случва като се запазват последните действия на модела в паметта. При учене, моделът използва извадка от пермутирация на миналите си действия, запазени в паметта. Това също помага да се премахнат времевите корелации между скоро извършени действия, и моделът да генерализира опита си.

3.2. Natural Language Understanding

За решаването на текстови проблеми, моделът трябва добре да разбира семантиката на подадения текст, за да може най - ефикасно да предприеме действие спрямо него. За да може да опишем семантиката на дадена думи или текст той се представя като вектор от числа, които го описват. Има изградени различни подходи, които се опитват да направят това.

Текущия най - добрия подход е това да се извърши с *word-embeddings.* Модела за това може да се използва готов пре-трениран на огромно количество думи, но може да се тренира на корпус за конкретния проблем. Друга възможност е да се вземе готов *word-embedding* и да продължи да се тренира върху конкретната задача. Има различни видове *word-embeddings.* Един подход за създаване на такива е *Word2vec,* който тренира модел за намиране на теглата на думите чрез изследване на зависимостта на дадена дума и контекста на думите покрай нея. Резултат от това е ефекта, че подобни вектори имат подобно семантично значение. Има и други модели като *Glove* (Pennington, Socher, Manning, 2014), които използват не само локални наблюдения, но и глобални такива. Подхода с *word-embeddings* е добър, когато се разглеждат думите поотделно, но когато искаме алгоритъма да разбира семантиката на цялото изречение и зависимостта между думите в него, те вече не се справят толкова добре.

LSTM (Hochreiter and Schmidhuber, 1997) е рекурентна невронна мрежа, която приема поредица от думи и намира вектори, които запазват информация за реда, който са подадени думите и по този начин разбира и зависимостта между тях. Един от най - големите проблеми на този модел е, че се тренира бавно, поради трудността му да се паралелизира. Друг проблем е това, че информацията за поредността на думите не дава много добра оценка за семантиката на думите.

Self-attention (Vaswani et al., 2017) се използва, за да се научи зависимостта между една дума в документ и всяка друга в същия документ. Така се изграждат семантични връзките между думи, които заедно могат да имат различен смисъл.

Transformers (Vaswani et al., 2017) са модели, които имат два основни компоненти - кодираща и декодираща. Кодирането на входа се прави чрез “глави”, които са слоеве, изчисляващи *self-attention*(Vaswani et al., 2017)*,* върху позиционно кодиран вход. Позиционното кодиране е нужно, защото *self-attention* слоя*,* сам по себе си няма разбиране за реда на думите.

BERT (Devlin et al., 2018) е метод за претрениране на езиков модел, идващ с вече трениран модел. Той се състои от двупосочна кодираща компонента от *Transformer* (Vaswani et al., 2017l)*.* Целта му е да намери векторно представяне на изречение, което да обхваща както синтактичното, така и семантичното значение. Поради това че моделът е вече трениран, може директно да се използва, но е задачи, за които е обучен са две конкретни - разпознаване на липсващи думи и класификация дали реда на две изречения е семантично правилен. Така първите слоеве се научават да разпознават синтактичната структура на изреченията, докато в последните слоеве семантиката. Ако задачата, за която се използва *BERT* не е подобна тези, за които е трениран, директно преизползване на модела може да доведе до лошо поведение. Препоръка от създателите на *BERT* е да се вземат теглата не от последния слой, а от някой от предишните, като добър избор е предпоследния.

4. Методи

4.1. Базов алгоритъм

Преди да започне, играта се инициализира. След това, при всяко изпълнение на действие се изпълняват следните стъпки:

1) Получаване на данни за текущото състояние на средата
2) Обработка на данните
3) Намиране на латентно представяне на данните
4) Получаване на оценка от подаденото представяне
5) Използване на оценката за генериране на команда
6) Изпълняваме от 1) до 6) до приключване на играта или достигане на максимума допуснати хода

Възможните данни, които могат да се получат на всеки ход са следните:

- Текст описващ промяната на състоянието в следствие от изпълнението на последната команда
- Оценка от последната команда
- Дали играта е приключила
- Допълнителна информация

В зависимост какво присъства в допълнителната информация се слага различно наказание, както е описано горе. Всички данни със съответното наказание са:

- Текстово описание на стаята, в която се намираме - Handicap 1
- Списък с всички предмети, с които разполагаме - Handicap 1
- Максималния брой точки, до които можем да стигнем -Handicap 0
- Целта на текущата игра (няма значение винаги е за готвене) - Handicap 0
- Всички понятия в текущата игра - Handicap 3
- Всички глаголи, които могат да се използват в командите - Handicap 2
- Шаблони на командите в текущата игра - Handicap 2
- Всички възможни команди - Handicap 5
- Рецепта (поредица от стъпки, които трябва да изпълним) - Handicap 4
- Едно възможно решение на играта (Walkthrough)- може да не е оптимално - не може да се използва при тестване

При обработка на данните по дадената текстова информация се конкатенира със интервал. При намиране на латентното пространство се извършват операции, като токенизиране и обработка на входа, използване на *embeddings,* модели, които намират векторното представяне на целия текст.

Стойностите от векторното представяне се подават на пет паралелни, оценяващи слоя. Получения резултат представлява масив от пет стойности с матрици с размерност - броя пъти, които играем една игра едновременно (големина на бач) x броя думи в речника. Петте стойности отговарят за оценки на думи в речника: една за глагола, една прилагателните за първото съществително, за първото съществително,

за прилагателните за второто съществително и още една за второто съществително. Това е шаблон само за конкретната игра в състезанието. *TextWorld* (Côté et al, 2018) може да генерира и други видове игри с други видове команди.

При генериране на действието се определя командата спрямо оценките от миналта стъпка. В нашата имплементация, при инициализирането на играта се създават масив от пет маски. Избират се индексите от речника отговарящи на петте най - големи оценки, съответно за петте думи в командата. Отделно се взимат пет произволни индекси за думите в командата. Финалните индекси, които се използват, се избират на принципа на епсилон-алчност.

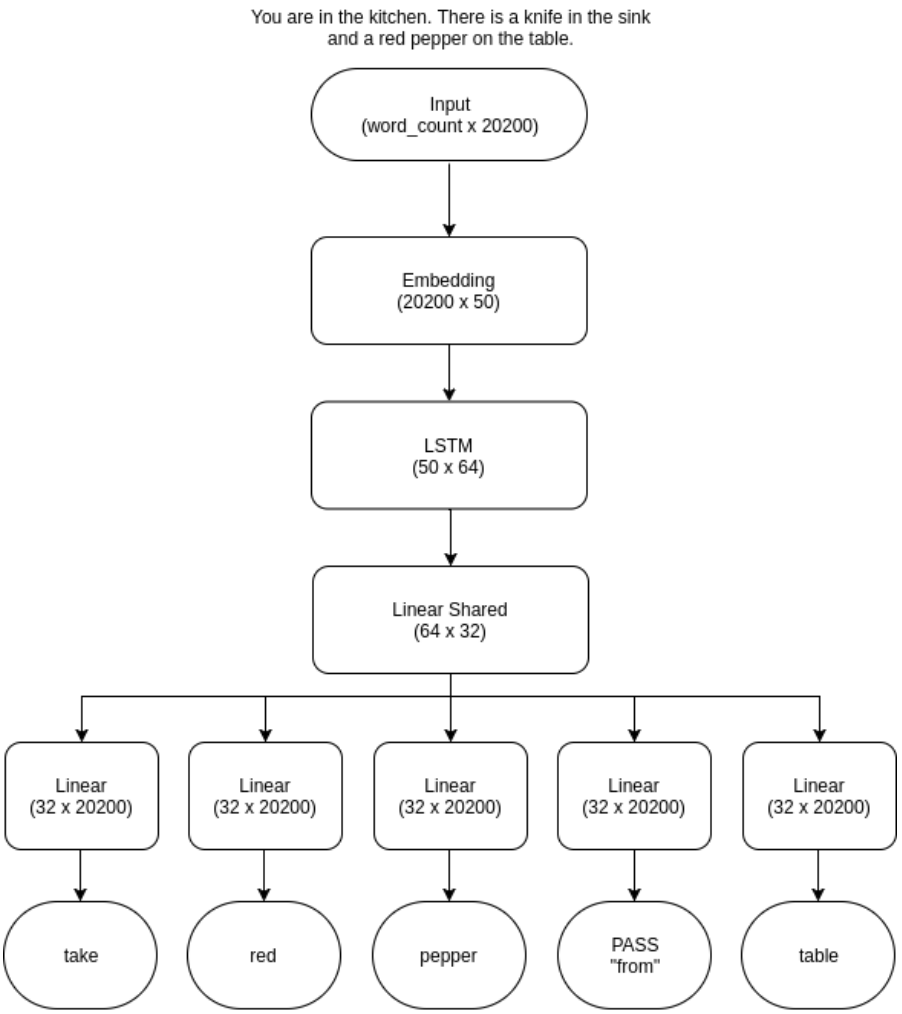
Епсилон-алчност (Sutton and Barto, 2018) се определя от хипер-параметър епсилон. Преди пускане на алгоритъма му се задава начална стойност и на колко стъпки с колко да се намалява. Той представлява вероятността да изберем произволна стъпка. Идеята му е в началото на играта, когато агента не знае нищо за пространството, да се избира по - често произволна команда. Това е нужно, защото агента трябва да изследва пространството от действия, за да открие по - голям диапазон от действия. След продължително играене, епсилон се намаля, за да може агентът да приложи наученото.

4.2. LSTM-DQN

От състезанието ни е предоставен начален модел (фиг. 2). Той реализира принципа на *DQN* обясенен горе. При него се зарежда речника за всички игри. Той е предоставен предварително за състезанието. Речника съдържа специалната дума ‘PASS’. Тя служи като възможност моделът да избере да пропусне излишна дума за командата.

При инициализация на играта глаголите и понятията от допълнителната информация се използват, за да се създадат пет маски. Маските отговарят кои думи в речника отговарят съответно на глаголите, прилагателните и съществителните. Четвъртата и петата маска са дубликати на първата и втората. Те отговарят за вторите прилагателни и съществителни и служат за улеснение на имплементацията. Частта, която намира латентното представяне (representation generator), е представена по следният начин. Има слой от ембединги, които се тренират от нулата. След тях следва *LSTM* (Hochreiter and Schmidhuber, 1997) слой.

Първата оптимизация, която правим е да сменим този ембединг слой с *Glove (Pennington, Socher, Manning,)* ембединги.



Фиг. 2 - Архитектура на LSTM DQN

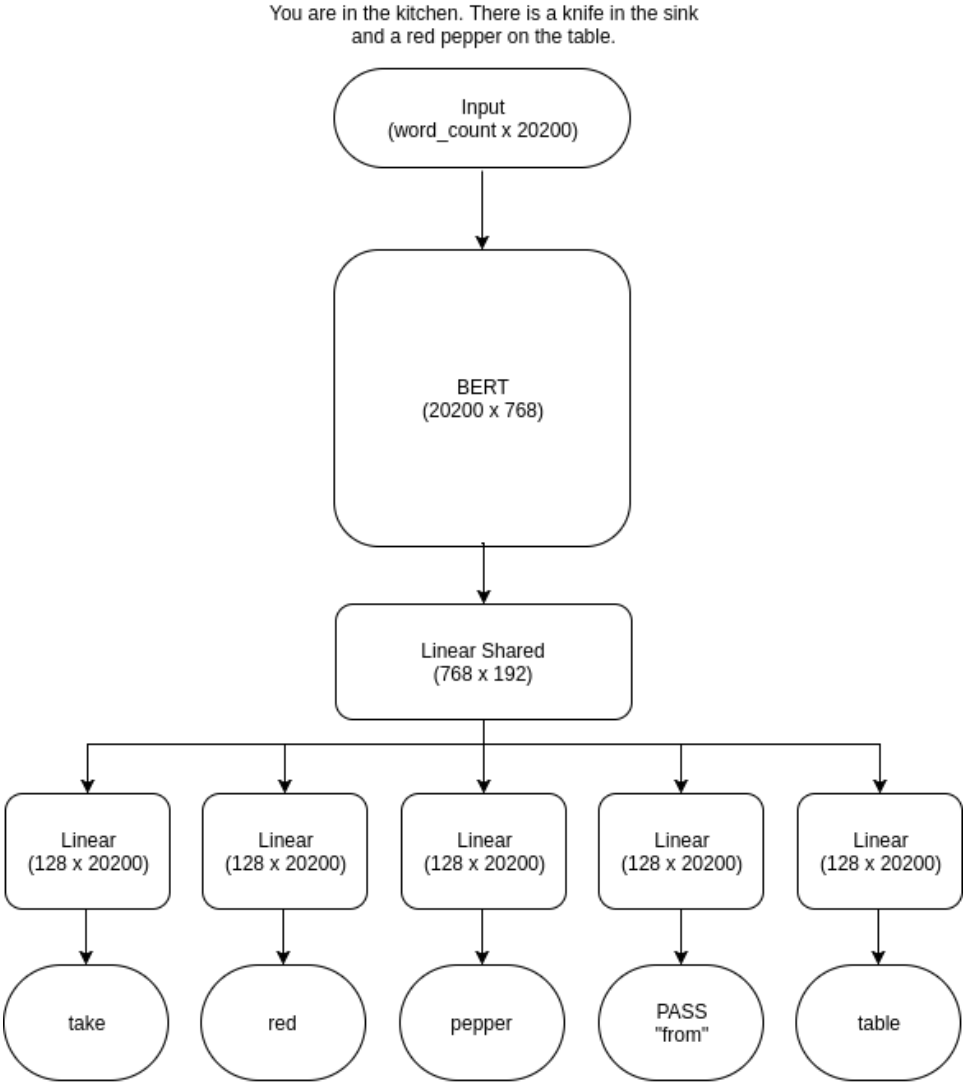
Всеки агент има памет, в която се пазят история на предишните действия.След всяко изпълняване на действие записваме в паметта:

- Състоянието, в което е била средата
- Действието, което е извършено
- Новото състояние, до което е довело действието
- Наградата, получена от средата за изпълненото действие

Действията се предприемат без мрежата да обновява теглата си. Обновяването на теглага се случва през k хода, което е хипер-параметър. При него се прави извадка от историята с размер d, което пак е хипер-параметър. Приемаме в случая, че d е големина на бач и намираме оценките за тези ходове от историята. Използваме мрежата, за да дадем оценка за хода, който се готвим да изпълним. Разликата между оценките от историята и оценката от модела е грешката, която се пробваме да минимизираме.

4.3. BERT-DQN

Поради *BERT* (Devlin et al., 2018) много нашумя в последно време и решихме да го изпробваме. Заместваме слоя с ембединги и *LSTM* слоя с *BERT (Devlin et al)* (фиг. 2). Използваме тренирания ‘bert-base-uncased’. Той е по - малкия, с размерност 768, върху английски език. При подаване на входния текст използваме *BERT (Devlin et al)* токенизатор за преобразуване на текста в специален вид токени, които вече се подават на *BERT* (Devlin et al., 2018) модела. Той връща матрица с размерност броя на думите и 768 (размера на ембединга) за всяка дума. Намираме средната стойност за всички думи и получаваме вектор от 768 стойности, които представляват векторното представяне на входния текст.



Фиг. 1 - Архитектура на BERT DQN

4.4. Double DQN

Следващия стъпка, която сме предприели да сменим *DQN* с *Double DQN.* При нея имаме идентична невронна мрежа като архитектура, която оценява най-доброто действие избрано от главния модел. Невронната мрежа, която се използва за избиране на най-доброто действие, се нарича *Q-network*, а тази, която оценява действието, се нарича Target Network. *Q-network* се обучава по описания за *DQN* начин.

4.5. Допълнителни оптимизации

За допълнително стабилизиране на алгоритъма сме използвали няколко други техники:

- Фиксиране на модела за оценяване (Lillicrap et al., 2015)

Има проблем когато двете мрежи в *Double DQN* се обновяват независимо, и той е, че ако много често се променя *Target network* се получава нестабилност в грешката и рязка промяна в loss функцията. Затова *Target network* се фиксира и се обновява на определен брой хода, като това се случва като се копират теглата на *Q-network*. Броят ходове е хипер-параметър на агента.

- N-стъпково оценяване (Hernandez-Garcia and Sutton, 2019)

При всяко обновяване се използва наградата само от последният ход. Това е малко информация за оценката на общото поведение на алгоритъма. За това вместо само от последната стъпка, се използват наградите от миналите n стъпки (определя се чрез хипер-параметър). За да не им се дава еднакъв приоритет, наградите се представят на степен - колко хода от текущия са се случили. Понеже наградата е в интервала [0, 1], колкото по отдавна са се случили, толкова по - малко влияние оказват.

- Prioritized experience replay (Schaul et al., 2015)

Всеки път ако се взимат произволни действия от историята, важните действия може да не се изтеглят достатъчно често, което да възпрепятства модела да ги научи. Решението на този проблем е имплементацията на Prioritized experience replay (Schaul et al., 2015) наречена - Prioritized replay memory. Цялата история се разделя на две части - алфа и бета памет. В алфа седят събитията с по - голям голям приоритет, а в бета тези с по - малко. При вземане на извадка, спрямо хипер-параметър определяме каква част от извадката да е от приоритетната и каква част да не е. Приоритетно действие е такова, че при неговото изпълнение сме получили позитивна награда от играта.

- Noisy Networks (Fortunato et al., 2017)

Те са линейни невронни мрежи с добавен параметричен шум към теглата им, и се използват за подпомагане на ефективното проучване на средата. Параметрите на шума могат да се изучават чрез градиентно спускане заедно с останалите тегла, но могат и да са фиксирани. Noisy Networks заместват конвенционални евристики за проучване като епсилон-алчност, и се справят с това по-ефикасно.

- Учене чрез имитиране

При RL методите се агента сам трябва да изследва пространството, в което се намира и сам да научи кои действия са правилни. В някои случаи това става много бавна и трудоемка задача. За това се използва специален вид трениране - учене чрез имитиране. Като вход да се подават, освен текущото състояние и демонстрация за решаване на проблема. При обновяване на теглата трябва да се оптимизира поведението на агента да работи максимално еднакво колкото демонстрацията. От състезанието разполагаме с едно решение на играта, което сме използваме за такъв вид обучение. С някаква вероятност се избира дали ще играем дадена игра, като демонстрация. При всеки ход се запомня в историята правилната стъпка, която се взема от готовото решение и я избираме за правилна стъпка. Нейната роля идва при обновяване на теглата. Когато се избират произволни действия от историята на базата, на които ще се обнови мрежата. Чрез демонстрационните ходове имаме повече положителни действия с голяма награда, от които да се учи мрежата. Проблема на този подход е, че не се демонстрира как да се стигне до избора на изкуствено добавените добри ходове и мрежата не може да извлече много полезна информация от тях.

5. Експерименти

Игрите, върху които тренирахме различните си агентите, са 10, избрани от създателите на състезанието. Те варират от такива, с малко ходове и прости команди, до такива, с 10 максимален брой положителни хода, нужни за победа, и команди, запълващи показаните горе шаблони.

Общи хипер-параметри за всички модели:

- Batch_size

В конкретната задача batch size представлява броя пъти, за които една игра се играе едновременно. Разликата в промяната му единствено помага за ускоряване на тренирането.

- Epsilon

Параметри за контрол на изследването на пространство чрез епсилон-алчност. Представлява шанс за вземане на случайно решение при трениране на агента. В началото на играта, когато агента не знае нищо за пространството, параметъра е 1, за да изследва пространството от възможни действия. Впоследствие, епсилон намалява всеки епизод, докато стигне стойност 0,2, в обозначен епизод. При трениране, за последния епизод използвахме номер 200, за да има време моделът да разучи пространството.

- Bert_model

Името на предварително тренирания модел, който използваме.

- Train_bert

Това е хипер-параметър, който определя дали да се тренират теглата на BERT модела Заради огромният брой параметри на bert и включително на другата част от мрежата не ни достигнаха ресурсите за да го включим.

- Imitate_rate

Определя с каква вероятност един бач от една игра да се изиграят чрез имитиране на предварително дефинирани действия. Моделите, които тренирахме с този хипер-параметър имаха много лошо поведение, затова той е изключен в експериментите, които сме описали по - долу.

- Imitate_first_epoch

Поради лошото поведение на imitate_rate, използвахме imitate_first_epoch, за да използваме имитация само първата игра за всички игри. Целта е да се добавят положителни действия за всяка игра още в началото, без да пречи в тренирането.. Този подход не подобри почти нищо, затова и него сме го изключили.

- replay_memory_capacity

Максимален размер на историята. Първоначален размер от 1 000 000, не може да се побере в паметта. Компромисен вариант, който не наврежда на запомнянето на агента, е 10 000.

- replay_memory_priority_fraction

Определя каква част от извадката при обновяване на теглата да съдържа отрицателни действия от историята, а остатъка са положителни. Избрали сме го да е 0.5, за да има равно тегло между положителни и отрицателни примери.

- update_per_k_game_steps

Използва се за определянето на колко хода да се обновяват теглата на мрежата. Колкото по - малък е този параметър, толкова по - често ще се тренира, за сметка на скоростта на трениране. Избрали сме тя да е 8, за да не забавяме прекалено много тренирането.

- replay_batch_size

Колко голяма извадка да се взима от историята при обновяване на теглата. С replay_memory_priority_fraction се определя каква част от тези да са положителни и каква да са отрицателни.

- noisy_std

Стандартното отклонение на шума в слоевете с Noisy Networks. Изпробвахме с 0.1, но в един момент не е достатъчно голям за да направи по - разнообразни команди в началото. За това сме го избрали да е 0.3, което подобри скоростта на изследването на областта.

- embedding_size

Размер на Embeddings слоя в starting-kit модела

- encoder_rnn_hidden_size

Размер на латентния слой намиращ векторната репрезентация на входа

- action_scorer_hidden_dim

Размера на всеки слой за намиране на оценка за всяка от целевите думи

- max_steps

Максимален брой ходове, за които се игра една игра. Оставихме го стандартно 100, защото това са максималният брой стъпки давани в състезанието за завършване на най - сложнте игри.

- Nsteps

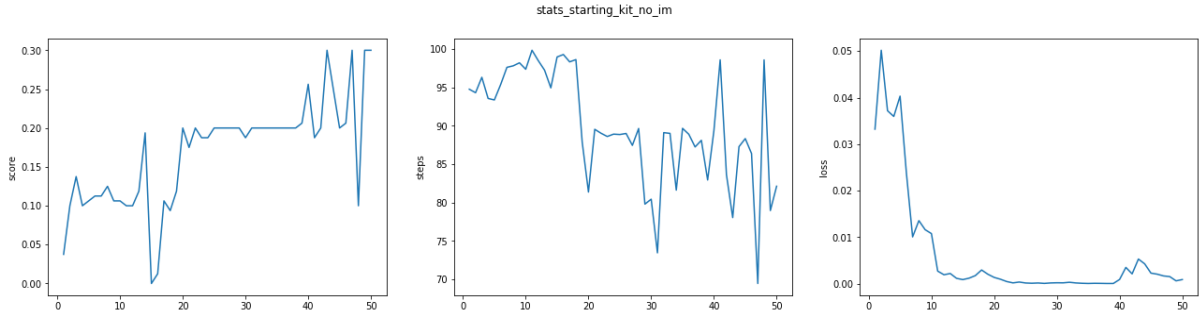
Брой стъпки назад, за които получаваме информация за наградите при N-стъпковото оценяване.

Други параметри на мрежата са: optimizer, learning rate, gradient clipping, dropout chance.
Единственият оптимизатор, който използваме е Adam, learning rate, вариращ от 0,001 до 0,01. Използваме gradient clipping (Pascanu, Mikolov, Bengio, 2013) в LSTM, за да избегнем прекалено голям градиент. Експериментирахме с dropout (Srivastava, 2014), от 0,1 до 0,3, но това не доведе до подобрение.

Агенти, използвани в експериментите, със специфичните им хипер-параметри:
(Ако хипер-параметър не е описан в моделите, се предполага, че е използвана стойност по премълчаване, описана горе)

Staring kit

Embedding_size: 32
encoder_rnn_hidden_size : 192
Action_scorer_hidden_dim: 64
epsilon_anneal_episodes: 100
epsilon_anneal_from: 1
epsilon_anneal_to: 0.2



BERT-DQN

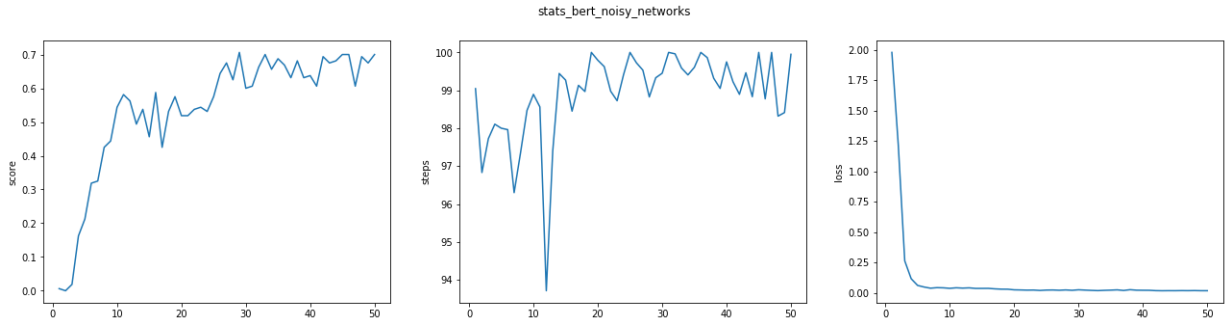
Embedding_size: 768
encoder_rnn_hidden_size : 192
Action_scorer_hidden_dim: 64
epsilon_anneal_episodes: 60
epsilon_anneal_from: 1
epsilon_anneal_to: 0.2
bert_model: 'bert-base-uncased'

Поради технически проблеми не е трениран достатъчно. Само 16 епохи.

BERT-DQN Noisy Networks

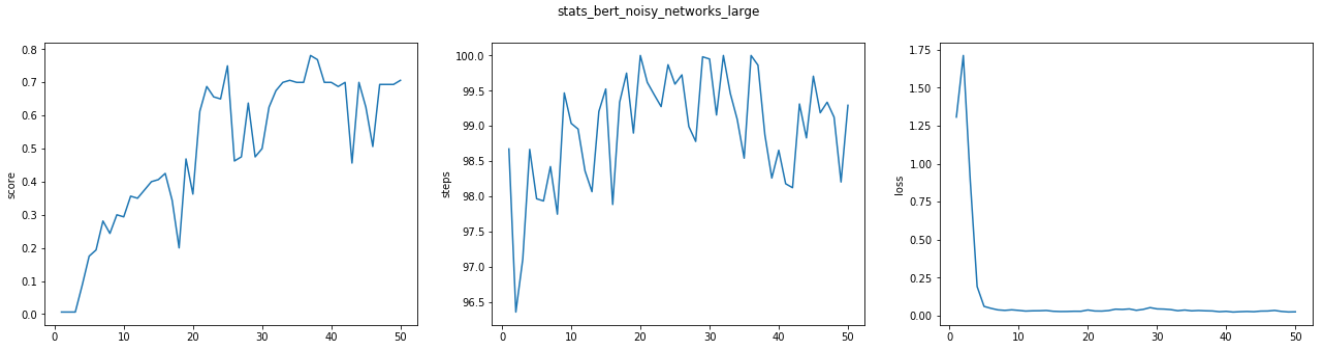
Embedding_size: 768
encoder_rnn_hidden_size : 192

Action_scorer_hidden_dim: 64
epsilon_anneal_episodes: 200
epsilon_anneal_from: 1
epsilon_anneal_to: 0.2
bert_model: 'bert-base-uncased'



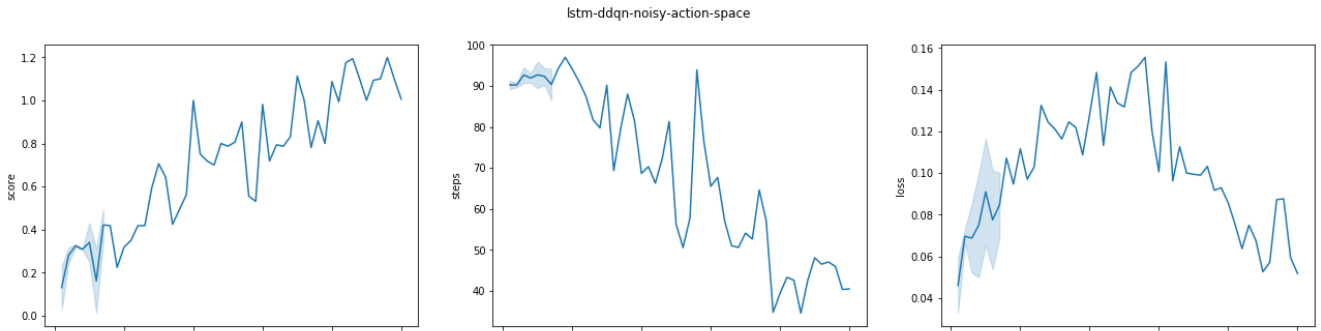
BERT noisy networks large

Embedding_size: 768
encoder_rnn_hidden_size : 384
Action_scorer_hidden_dim: 128
epsilon_anneal_episodes: 200
epsilon_anneal_from: 1
epsilon_anneal_to: 0.2
bert_model: 'bert-base-uncased'



LSTM-DDQN Noisy networks with smaller action space

Embedding_size: 50
encoder_rnn_hidden_size : [64, 64]
Action_scorer_hidden_dim: 128
epsilon_anneal_episodes: 200
epsilon_anneal_from: 1
epsilon_anneal_to: 0.2
noisy_std: 0.3



Оценка

При оценка на моделите сме избрали извадка от съдържаща десет игри от всички игри за трениране. Игрите са сравнително лесни. Причината за това е, че моделите ни се представят слабо и сме подбрали така игрите за да се види разликата в поведението на моделите. Оценките са много оптимистични поради няколко причини. Както споменахме, те са сравнително лесни. Те са от данните дадени за трениране. При оценяване от състезанието всички игри са различни от тези, с които разполагаме.

Модели	Сума награди ↑	Среден брой стъпки за приключване на игра ↓	Обучение
starting-kit	30	90	100 епохи по 16 големина на бач
Bert DQN	50	100	16 епохи по 16 големина на бач
BERT-DQN Noisy Networks	70	100	80 епохи по 16 големина на бач
BERT-DQN Noisy Networks Large	90	100	80 епохи по 16 големина на бач
LSTM-DDQN Noisy networks with reduced action space	140	32	100 епохи по 10 големина на бач

Моделът, който постига най - много точки не е със сигурност най - добър. При него сме използвали различен речник съдържащ само думите, които могат да се съдържат и в командите от всички игри за трениране. Тоест при реално оценяване може модела да не познава много от тези понятия. Той е най - оптимистичния модел.

Използвани технологии, платформи и библиотеки

- **pandas** - зареждане и обработка на данни
- **matplotlib -** визуализация на графики
- **numpy** - работа с многомерни масиви
- **spacy** - токенизатор в началния модел
- **gym** - взаимодействие между агент и игра
- **textworld** - зареждане на игри, генерирани от състезанието
- **tqdm** - по - структурирано принтиране на статистики при трениране
- **pytorch** - имплементиране на моделите
- **pytorch_pretrained_bert** - имплементация на *BERT (Devlin et al)* на pytorch и токенизатор за *BERT*
- **google colab** - за трениране и експерименти с моделите.

6. Заключение

Избрахме този проект поради важността му за развитието на разбирането на език и игровия подход, който се прилага за това. Решението на този проблем изисква имплементация на много различни подходи, като разбиране на текст, запомняне на важна информация, следване на цел и генерализация за много различни среди. Въпреки че все още сме далеч от човешкия резултат, получените резултатите са добро начало за по-нататъчно развитие на задачите с играене на текстови игри. Моделът се справя с прости и средно комплексни игри, но изпитва трудности в по-сложни сценарии.

Част от идеите за усъвършенстване, които не успяхме да реализираме, са:

Подобряване на обучение чрез имитация. Нашата имплементация не подобрява алгоритъмът. Напротив може да се каже, че го залъгва. При имплементирането на трениране чрез имитация трябва да се представят теглата в мрежата, за които би се получила демонстративната стъпка и да се опитаме да научим алгоритъма как да достига сам до нея вместо да я използва наготово.

Трениране на агентът на бачове от игри, започващи от по-лесни и стигащи до по-трудоемки. Това би подобрило генерализацията на модела, като първо научи основните обекти и цели от простите игри, и в последствие прилага наученото и оптимизира стратегии за по-комплексните.

Паметта на агента също може да претърпи усъвършенстване. Начин за това са *Memory Networks* (Weston, Chopra, Bordes, 2014), предназначени да решават проблема за изучаване на дългосрочни зависимости в последователни данни, чрез предоставяне на изрично представяне на паметта за всеки токен в последователността.

Deep Recurrent Q-Network (Hausknecht, Stone, 2014) е подход за справяне с частичната видимост на предишни ходове. Идеята е да се използва една рекурента клетка вместо общия линеен слой в мрежата. Така на всяка стъпка рекурента клетката предава информация на същата клетка в следващата времева стъпка, запазвайки контекста от предишни ходове.

7. Принос

- **Мартин**

Приносът се състои в търсене, четене и разбиране на документация за всички части на проекта. Интегриране на *BERT* тестване и експерименти с него. Обработка на

входните данни и инициализация преди трениране. Опити с намаляване на пространството от възможни команди. Имплементиране на трениране чрез имитация.

- **Николай**

Четене и разбиране на алгоритми свързани с обучение чрез подсилване. Запознаване и работа с *PyTorch*. Запознаване и използване на *Spacy.* Имплементиране на *Double DQN* алгоритъма, подобрения като *Fixed targets, Prioritized Replay Memory*, *Noisy Networks* експерименти с архитектурата. Интегриране на *Glove.*

8. Библиография

[Côté et al, 2018] Côté, Marc-Alexandre, et al. "Textworld: A learning environment for text-based games." *arXiv preprint arXiv:1806.11532* (2018).

[Sutton and Barto, 2018] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Puterman, 1994] Martin L. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming.John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[Mnih et al., 2013] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

[van Hasselt et al., 2016] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[Sutton, 1988] Sutton, Richard S. "Learning to predict by the methods of temporal differences." *Machine learning* 3.1 (1988): 9-44.

[Lin, 1992] Lin, Long-Ji. "Self-improving reactive agents based on reinforcement learning, planning and teaching." *Machine learning* 8.3-4 (1992): 293-321.

[Hochreiter and Schmidhuber, 1997] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

[Vaswani et al., 2017] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

[Devlin et al., 2018] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[Schaul et al., 2015] Schaul, Tom, et al. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952* (2015).

[Hernandez-Garcia and Sutton, 2019] Hernandez-Garcia, J. Fernando, and Richard S. Sutton. "Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target." *arXiv preprint arXiv:1901.07510* (2019).

[Lillicrap et al., 2015] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

[Fortunato et al., 2017] Fortunato, Meire, et al. "Noisy networks for exploration." *arXiv preprint arXiv:1706.10295* (2017).

[Weston, Chopra, Bordes, 2014] Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." *arXiv preprint arXiv:1410.3916* (2014).

[Hausknecht, Stone, 2014]Hausknecht, Matthew, and Peter Stone. "Deep recurrent q-learning for partially observable mdps." *2015 AAAI Fall Symposium Series*. 2015.

[Pennington, Socher, Manning, 2014] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

[Pascanu, Mikolov, Bengio, 2013] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." *International conference on machine learning*. 2013.

[Srivastava, 2014] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15.1 (2014): 1929-1958.

“PyTorch Documentation” *Pytorch Documentation - PyTorch Master Documentation*, https://pytorch.org/docs/stable/index.html.

"TextWorld." *Microsoft Research*, 2018,
www.microsoft.com/en-us/research/project/textworld/.