IA353A - Neural Networks
EFC4

Rafael Claro Ito
(R.A.: 118430)

July 2020

Question 8: RL

The Jupyter notebook related to this section with the results presented here can be opened in Google Colab environment with following link:
https://drive.google.com/file/d/1Xya6E4BgNVOPlzPRKbb3C59rvKI6V2nk/view?usp=sharing

8.1 Training results and two study cases

8.1.1 Maze 1

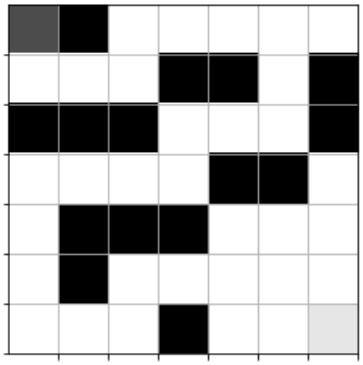
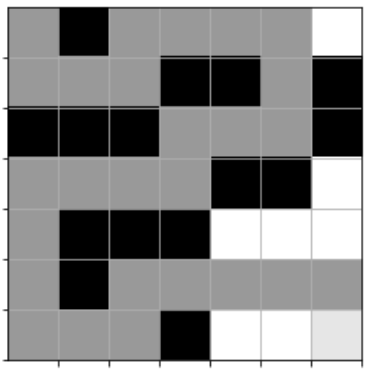


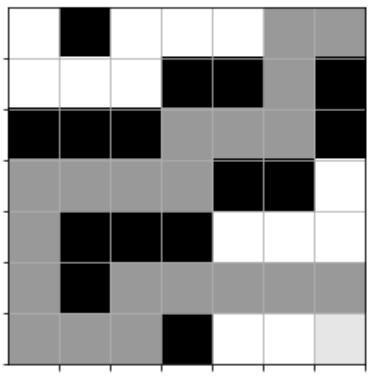
Figure 2: Maze 1

Epoch	Loss	Episodes	Win count	Win rate	Time
0	0.0317	51	1	0.000	2.7 s
10	0.0766	109	5	0.000	45.6 s
20	0.0023	102	10	0.000	77.8 s
30	0.0012	6	17	0.625	97.6 s
40	0.0627	103	24	0.667	117.0 s
50	0.0010	6	30	0.708	143.0 s
60	0.0025	4	39	0.708	152.4 s
70	0.0065	102	46	0.750	176.5 s
80	0.0019	2	56	0.833	188.0 s
90	0.0013	23	66	0.958	198.4 s
94	0.0010	21	70	1.000	199.8 s

Table 1: Training results summary for maze 1



(a) Maze 1 - Study case 1



(b) Maze 1 - Study case 2

8.1.2 Maze 2

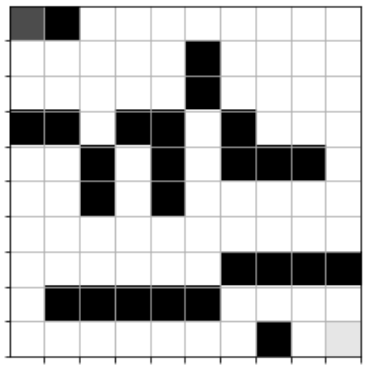
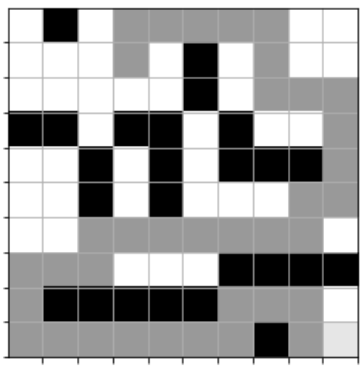


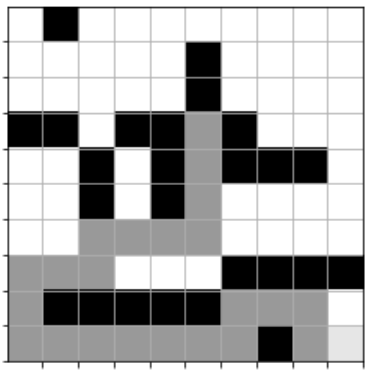
Figure 4: Maze 2

Epoch	Loss	Episodes	Win count	Win rate	Time
0	0.0856	147	1	0.000	8.8 s
10	0.0011	219	4	0.000	128.4 s
20	0.0042	55	6	0.000	237.7 s
30	0.0041	21	13	0.000	297.6 s
40	0.0044	67	22	0.000	346.0 s
50	0.0026	30	31	0.600	380.5 s
60	0.0043	23	41	0.740	6.69 min
70	0.0006	18	51	0.900	7.00 min
80	0.0071	35	61	0.960	7.44 min
90	0.0005	18	71	0.980	7.67 min
100	0.0015	4	81	1.000	7.92 min
110	0.0010	11	91	1.000	8.16 min
120	0.0008	18	101	1.000	8.43 min
130	0.0004	19	111	1.000	8.84 min
140	0.0012	20	121	1.000	9.13 min
150	0.0007	34	131	1.000	9.54 min
160	0.0013	61	141	1.000	9.83 min
165	0.0002	28	146	1.000	9.99 min

Table 2: Training results summary for maze 2



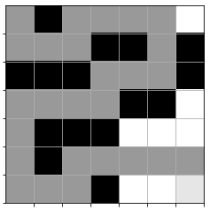
(a) Maze 2 - Study case 1



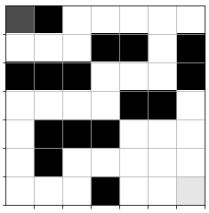
(b) Maze 2 - Study case 2

8.2 Q-value for three different states

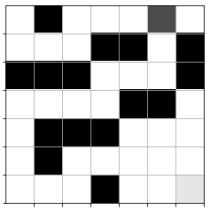
8.2.1 Maze 1 - Study case 1



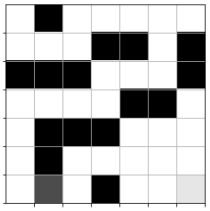
(a) Maze 1: Study case 1



(b) State 1
- left: -0.6266
- up: -0.4188
- right: -0.0743
- down: 0.0651

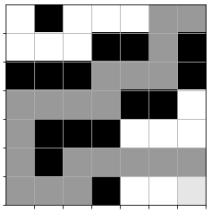


(c) State 2
- left: -0.4398
- up: -0.4012
- right: -0.2719
- down: -0.1840

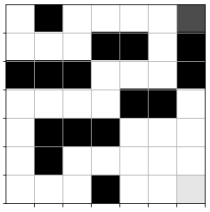


(d) State 3
- left: -0.0811
- up: -0.0670
- right: 0.4976
- down: 0.0538

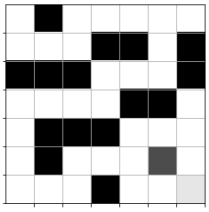
8.2.2 Maze 1 - Study case 2



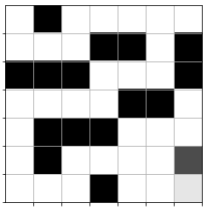
(a) Maze 1: Study case 2



(b) State 1
- left: -0.3038
- up: -0.4816
- right: -0.8406
- down: -0.4413

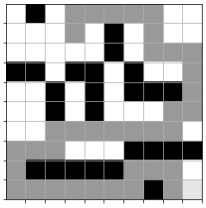


(c) State 2
- left: 0.3047
- up: 0.6444
- right: 0.9281
- down: 0.0919

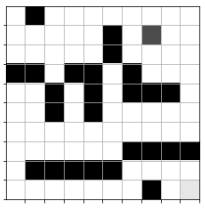


(d) State 3
- left: 0.3186
- up: 0.6454
- right: -1.9548
- down: 1.0120

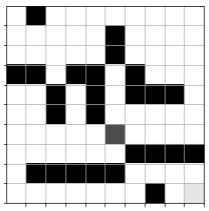
8.2.3 Maze 2 - Study case 1



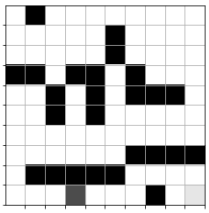
(a) Maze 1: Study case 1



(b) State 1
- left: -0.5747
- up: -0.4207
- right: -0.4067
- down: -0.3697

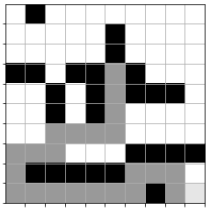


(c) State 2
- left: -0.2018
- up: -0.6692
- right: -0.5693
- down: -0.6134

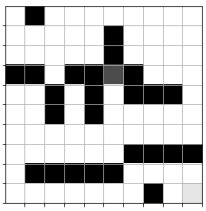


(d) State 3
- left: 0.1001
- up: -0.1958
- right: 0.4426
- down: -0.0384

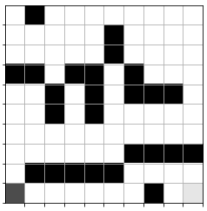
8.2.4 Maze 2 - Study case 2



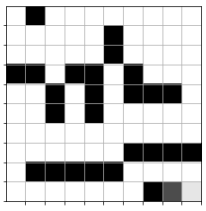
(a) Maze 2: Study case 2



(b) State 1
- left: -0.5848
- up: -0.6763
- right: -0.4808
- down: -0.2210



(c) State 2
- left: -0.0098
- up: -0.2776
- right: 0.2342
- down: 0.0314



(d) State 3
- left: -0.2766
- up: 0.0523
- right: 0.9958
- down: -0.1527

8.3 RMS during the training

Explique como é definida a função de erro quadrático médio usada no treinamento.

Inicialmente, devemos relembrar a equação de Bellman. Queremos achar a função Q-valor ótima, Q^*, que satisfaça a seguinte equação:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

A política ótima π^* é dada se tomarmos a melhor ação de acordo com os valores da função Q^*.

Para resolver a equação anterior, temos um algoritmo iterativo dado por:

$$Q_{i+1}(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Quando i tende a infinito, temos que Q_i tende a Q^*.

Entretanto, essa abordagem não é escalável. Assim o que se faz é usar uma rede neural para aprender e estimar a função Q-valor, sintetizando portanto, um mapeamento entre os estados e os q-valores.

Faremos então:

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Onde θ representa os parâmetros do modelo (pesos da rede neural).

Assim, tomando y_i como a saída da rede neural e $L_i(\theta_i)$ a função de perda usada durante seu treinamento, podemos retomar a equação de Bellman aproximando o lado direito com o lado esquerdo:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a'; \theta_{i-1}) | s, a \right]$$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho} \left[(y_i - Q(s,a;\theta_i))^2 \right]$$

E é justamente assim que é definida a função de erro quadrático médio usado no treinamento. Procura-se um mapeamento entre os estados, usado como entrada da rede neural (neste caso é o panorama do labirinto), e a a função Q-valor ótima. Ao minizarmos o erro quadrático médio entre a saída da rede e função Q-valor parametrizada por theta, estamos na verdade forçando que o lado esquerdo da equação de Bellman seja igual ao lado direito. Conforme o agente se movimenta e explora o ambiente, atua-se no vetor de pesos através de técnicas de gradiente, e com isso chegamos em um mapeamento onde a saída da rede neural representa a função Q-valor ótima aproximada.

Para finalizar, usando como gancho para a resposta do próximo item, temos que a entrada da rede (inputs) o estado do ambiente e a saída da rede (targets) calculada como a recompensa dada pela ação tomada anteriormente mais o fator de desconto (gamma) multiplicado pelo máximo de $Q(s', a')$ (que representa o máximo da saída do próximo estado, ou apenas a recompensa em caso de game over).

8.4 Experience replay

Explique como é trabalhada a técnica de experience replay.

Para se trabalhar com a técnica de experience replay, é proposta uma classe denominada Experience. Nesta classe temos o "construtor" da classe __init__ (magic method, ou ainda "dunder") e três métodos: remember, predict e get_data.

A principal ideia aqui é usar uma memória para armazenar episódios. Um episódio é composto de uma lista com cinco elementos:

- envstate: estado do ambiente contendo os valores de todas células do labirinto (em um array 1D).
- action: uma das quatro ações que o agente (rato) pode tomar (cima, baixo, direita ou esquerda).
- reward: recompensa recebida pela ação tomada.
- envstate_next: novo estado do ambiente após a ação do agente.
- game_over: valor booleano que indica se houve game over (win ou lose).

Assim, para cada movimento do rato, temos um episódio que podemos inserir em uma memória. Conforme essa memória vai enchendo, defini-se um limite para deletar episódios mais antigos armazenados. Neste caso em específico, adotou-se uma memória de tamanho padrão 1000, mas definindo-a como 8 vezes o tamanho do labirinto na chamada do treinamento da rede neural.

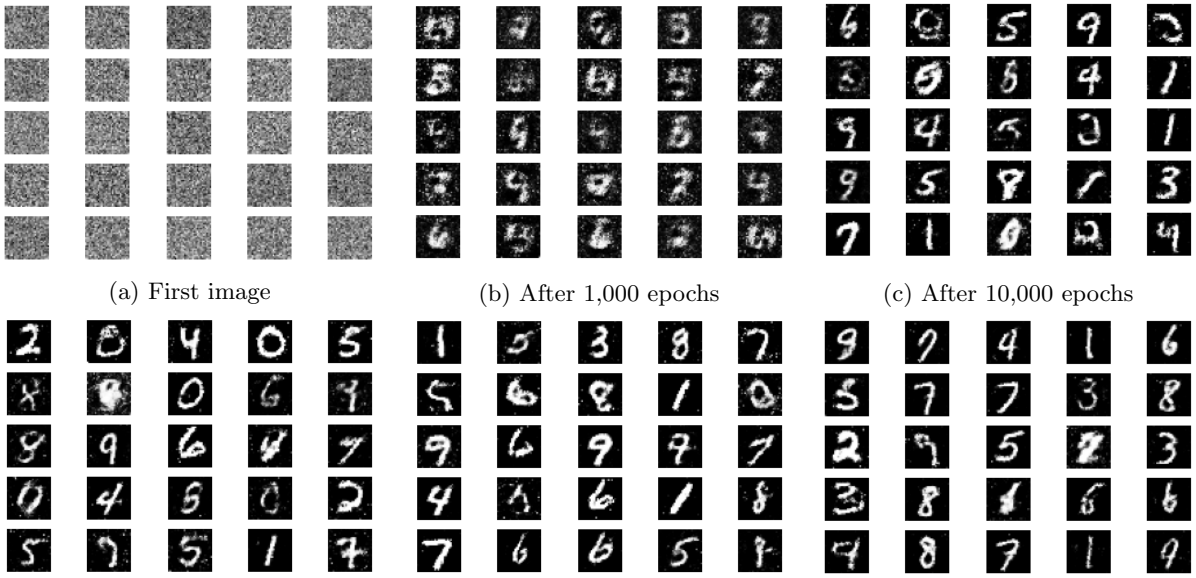
O método remember é chamado após cada movimento do agente, sendo sua função armazenar as informações daquele episódio na memória. O método get_data é usado para pegar da memória a entrada (inputs) e saída (targets) que será usado no treinamento da rede neural, sendo que o número de amostras retornado por get_data é definido como o mínimo entre o tamanho da memória e 50 (data_size). E é esse processo que caracteriza a técnica experience replay, armazenando experiências recentes do agente (com a premissa de quanto mais ele se movimenta no ambiente, melhor vai ficando a predição da rede neural e portanto seus próximos movimentos) e usando dessas experiências para o treinamento, dado que a solução do problema trata-se de um processo iterativo.

Inicialmente, a saída da rede neural produzirá resultados aleatórios, mas conforme o treinamento for avançando e os parâmetros ajustados adequadamente, sua saída vai convergindo para a solução da equação de Bellman.

9 Question 9: GAN

The Jupyter notebook related to this section with the results presented here can be opened in Google Colab environment with following link:
https://drive.google.com/file/d/1WWdc3M0UObMp1jVCcndmor6BTdLe18_e/view?usp=sharing

9.1 MNIST

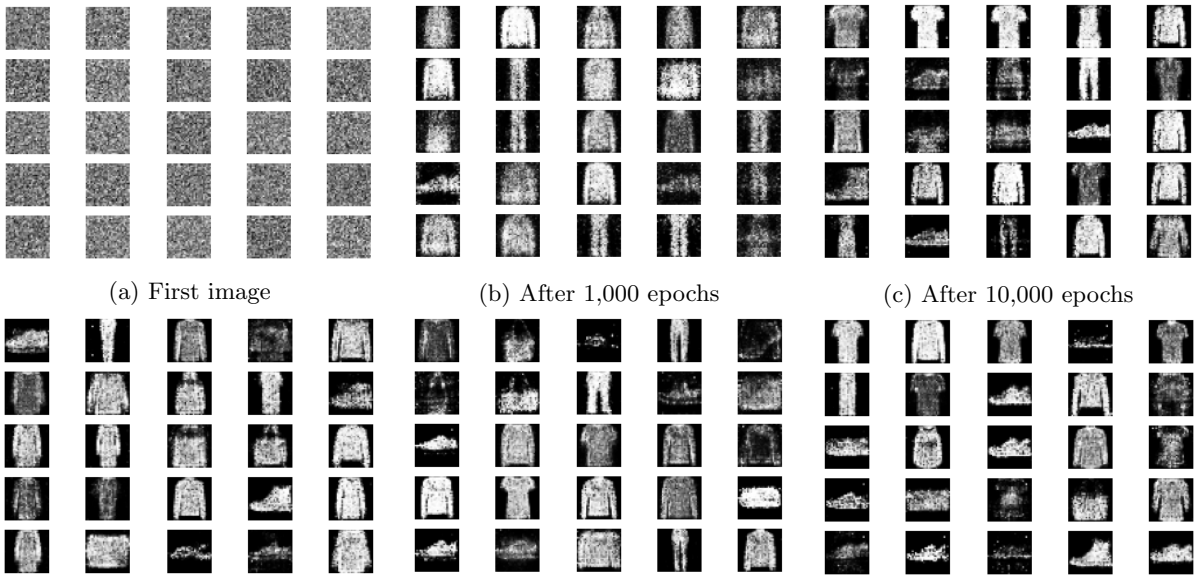


(a) First image (b) After 1,000 epochs (c) After 10,000 epochs

(d) After 20,000 epochs (e) After 30,000 epochs (f) After 50,000 epochs

Figure 10: GAN trained for MNIST dataset

9.2 Fashion MNIST



(a) First image (b) After 1,000 epochs (c) After 10,000 epochs

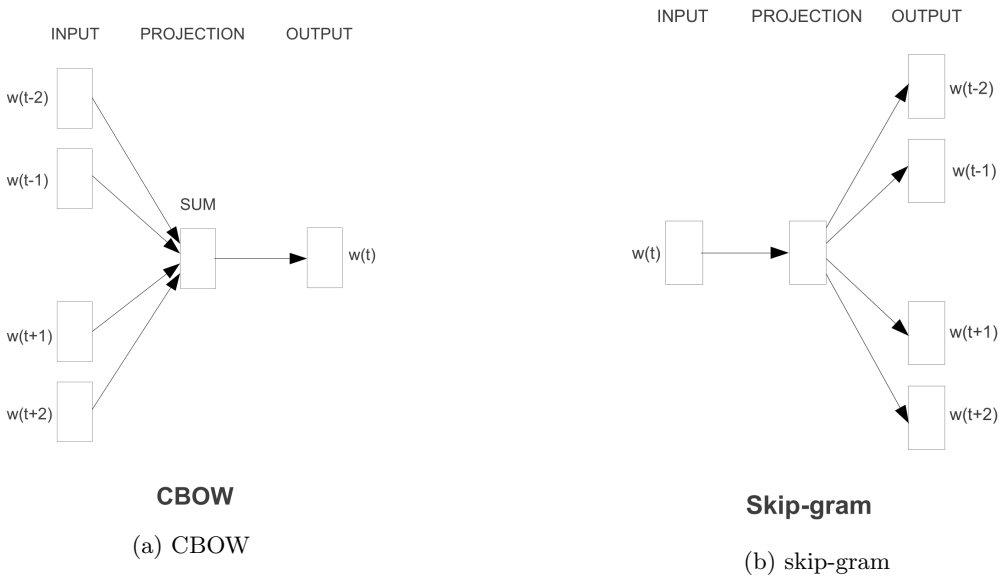
(d) After 20,000 epochs (e) After 30,000 epochs (f) After 50,000 epochs

Figure 11: GAN trained for fashion MNIST dataset

10 Question 10: NLP

The Jupyter notebook related to this section with the results presented here can be opened in Google Colab environment with following link:
https://drive.google.com/file/d/1EAU2toNRn-MjXyO_YAsZFejdaJDapibX/view?usp=sharing

10.1 word2vec



(a) CBOW

(b) skip-gram

Figure 12: Architectures used in word2vec

O word2vec, proposto por (Mikolov et al., 2013) é uma ferramenta que fornece uma implementação eficiente das arquiteturas CBOW (continuous bag-of-words) e skip-gram para computar representação de vetores de palavras. Um resumo de uma página deste artigo feito por mim pode ser encontrado no seguinte link. O código do wordvec está disponível em https://code.google.com/archive/p/word2vec/.

Resumidamente, o modelo CBOW tenta prever a palavra atual, baseada no contexto de algumas palavras anteriores e algumas palavras posteriores. Já o modelo Skip-gram tenta maximizar a classificação de uma palavra baseado em outra palavra na mesma frase.

Além dessas duas arquiteturas propostas, o artigo também apresenta medidas de similaridades sintáticas e semânticas entre palavras (feitas algebricamente), a criação do dataset "Semantic-Syntactic Word Relationship test set" e uma forma de treinamento paralelo implementada em um framework chamado "DistBelief".

Em termos práticos, o que se consegue com o word2vec é uma representação de palavras em um espaço de dimensão reduzida (um vetor no $\mathbb{R}^{300}$ por exemplo). Com isso, elimina-se a esparsidade da técnica anterior denominada bag-of-words, one tinha-se uma representação one-hot das palavras, isto é, a entrada da rede neural tinha o tamanho do vocabulário usado.

Esses vetores são denominados de word embeddings e são usados na grande maioria de arquiteturas SOTA (state-of-the-art). Por exemplo, na arquitetura BERT, usa-se três tipos de embeddings: embedding posicional, embedding de segmento e embedding de palavras.

Por fim, é interessante ressaltar as relações entre embeddings obtidas através de simples operações algébricas. Por exemplo, se considerarmos a relação entre embeddings de palavras de país e capital, temos a seguinte relação: Paris - France + Italy = Rome. Isto é, operando com os embeddings das palavras Paris, France e Italy, temos como resultado um vetor, sendo que a palavra mais próxima desse vetor resultante é o embedding da palavra Rome. Um outro exmplo clássico é a operação entres os vetores das palavra King - Man + Woman. O embedding mais próximo deste resultado é o da palavra Queen.

Um outro projeto semelhante é o GloVe, proposto em (Pennington et al., 2014), que também propõe vetores para representação de palavras. Um exemplo interessante pode ser encontrado na página inicial do projeto, onde é mostrado que as palavras cujos embedding são mais próximos do embedding da palavra frog são: frogs, toad, litoria, leptodactylidae, rana, lizard e eleutherodactylus.

10.2 t-SNE

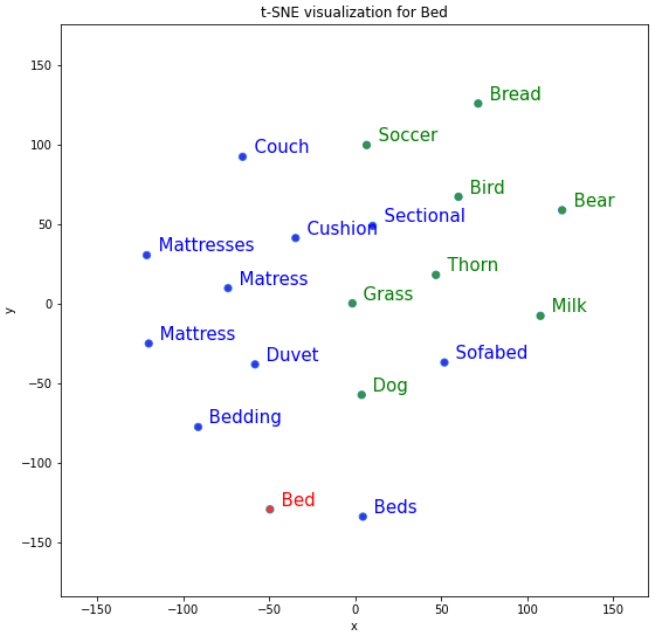


Figure 13: T-SNE 1

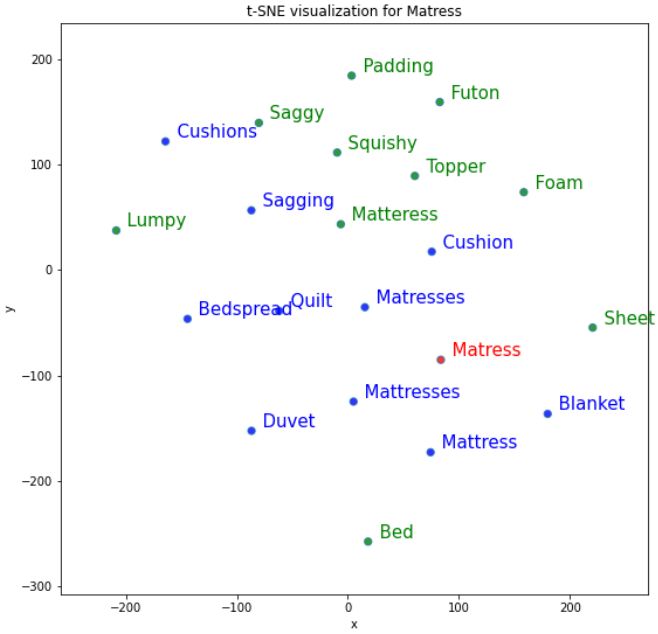


Figure 14: T-SNE 2

10.3 Results discussion