



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

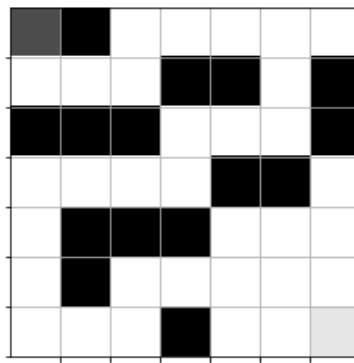
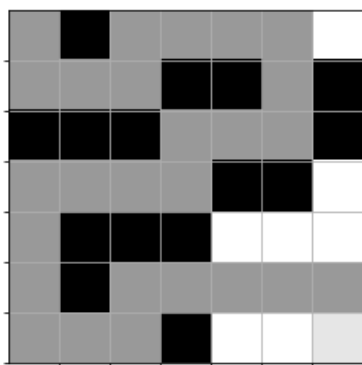


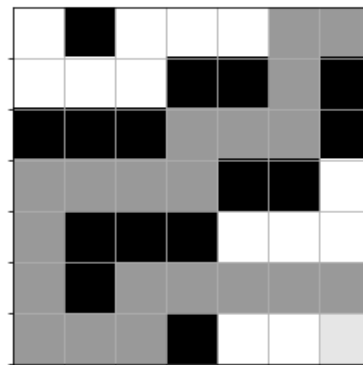
Figure 2: Maze 1

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

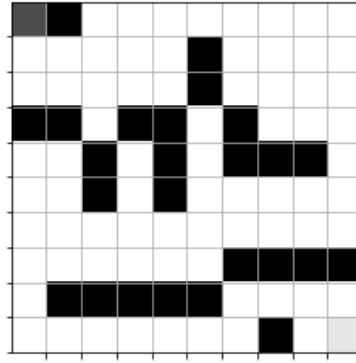
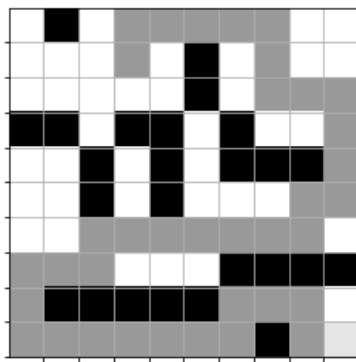


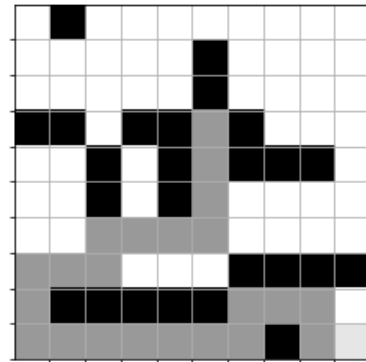
Figure 4: Maze 2

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



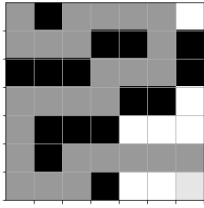
(a) Maze 2 - Study case 1



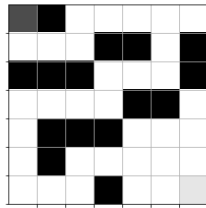
(b) Maze 2 - Study case 2

8.2 Q-value for three different states

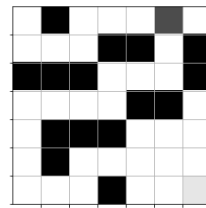
8.2.1 Maze 1 - Study case 1



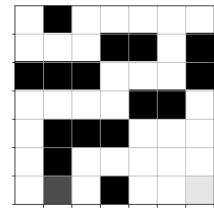
(a) Maze 1: Study case 1



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

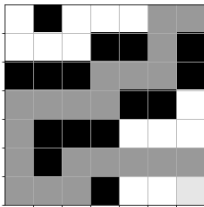


(c) State 2
 • left: -0.4398
 • up: -0.4012
 • right: -0.2719
 • down: -0.1840

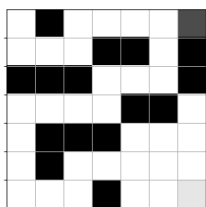


(d) State 3
 • left: -0.0811
 • up: -0.0670
 • right: 0.4976
 • down: 0.0538

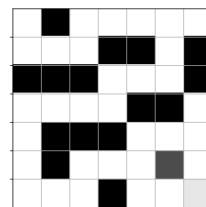
8.2.2 Maze 1 - Study case 2



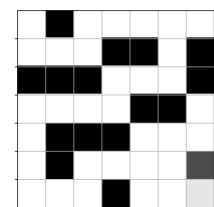
(a) Maze 1: Study case 2



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

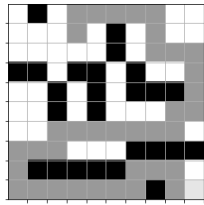


(c) State 2
 • left: 0.3047
 • up: 0.6444
 • right: 0.9281
 • down: 0.0919

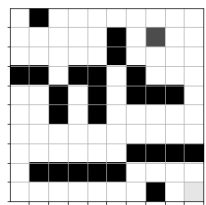


(d) State 3
 • left: 0.3186
 • up: 0.6454
 • right: -1.9548
 • down: 1.0120

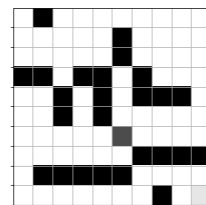
8.2.3 Maze 2 - Study case 1



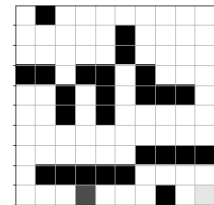
(a) Maze 2: Study case 1



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

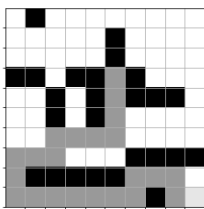


(c) State 2
 • left: -0.2018
 • up: -0.6692
 • right: -0.5693
 • down: -0.6134

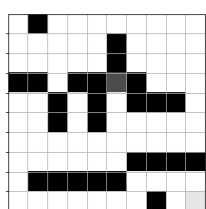


(d) State 3
 • left: 0.1001
 • up: -0.1958
 • right: 0.4426
 • down: -0.0384

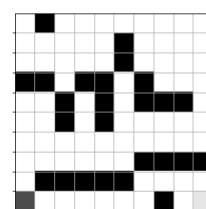
8.2.4 Maze 2 - Study case 2



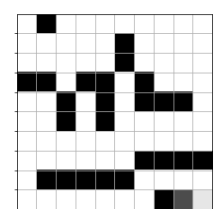
(a) Maze 2: Study case 2



(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 lembrar 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:

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

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

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`). 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

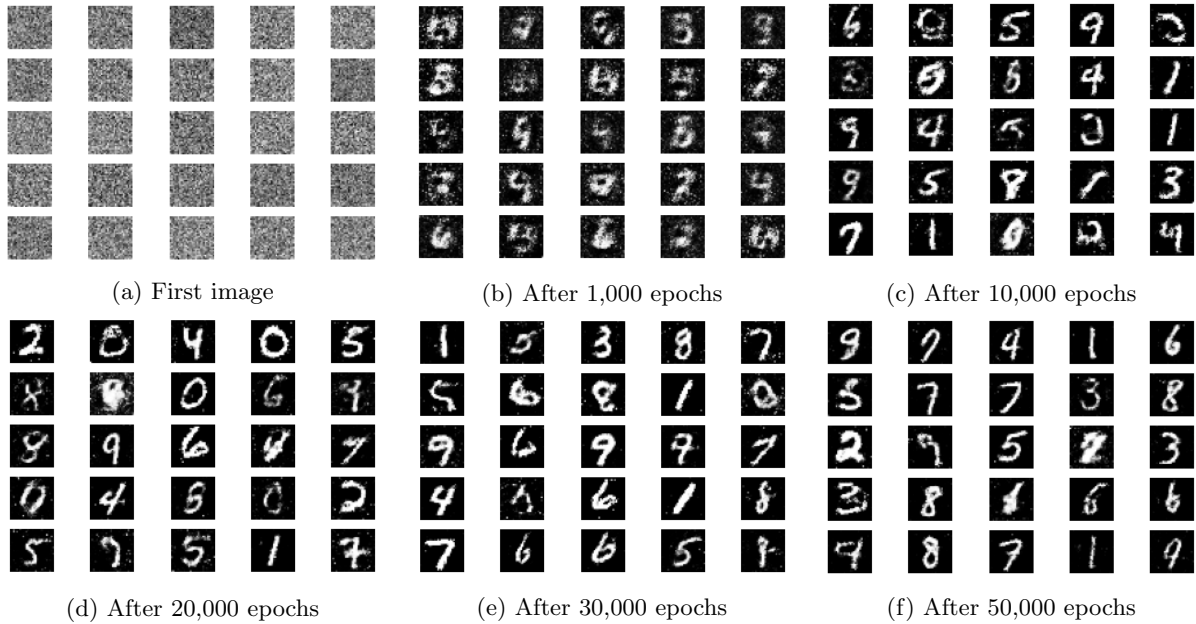


Figure 10: GAN trained for MNIST dataset

9.2 Fashion MNIST

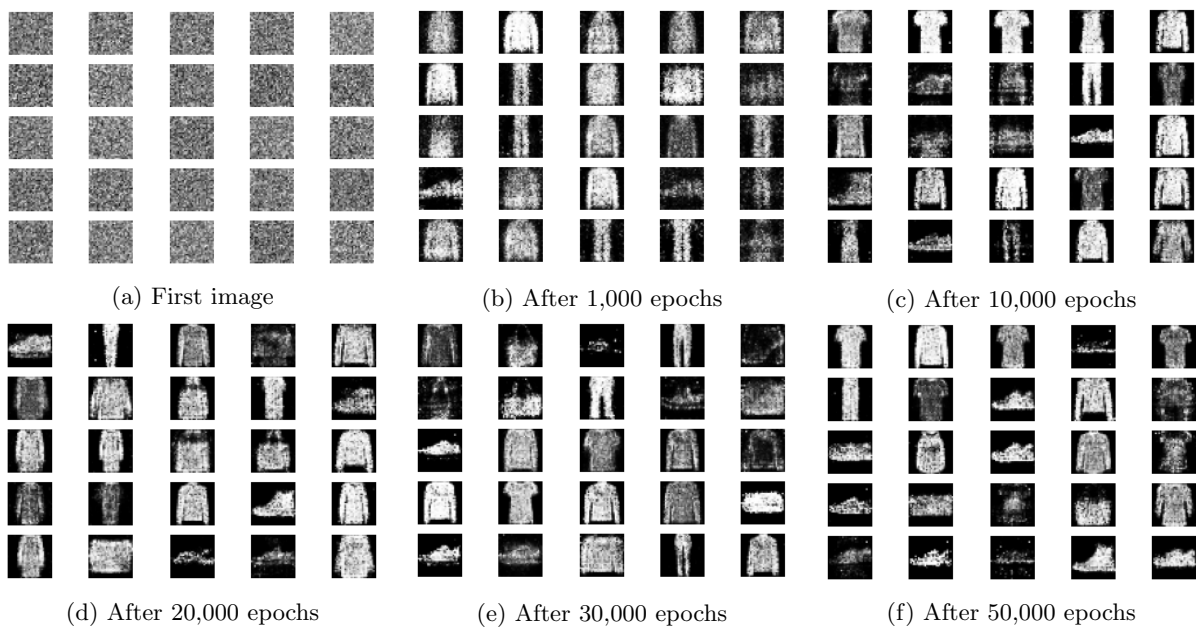


Figure 11: GAN trained for fashion MNIST dataset

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

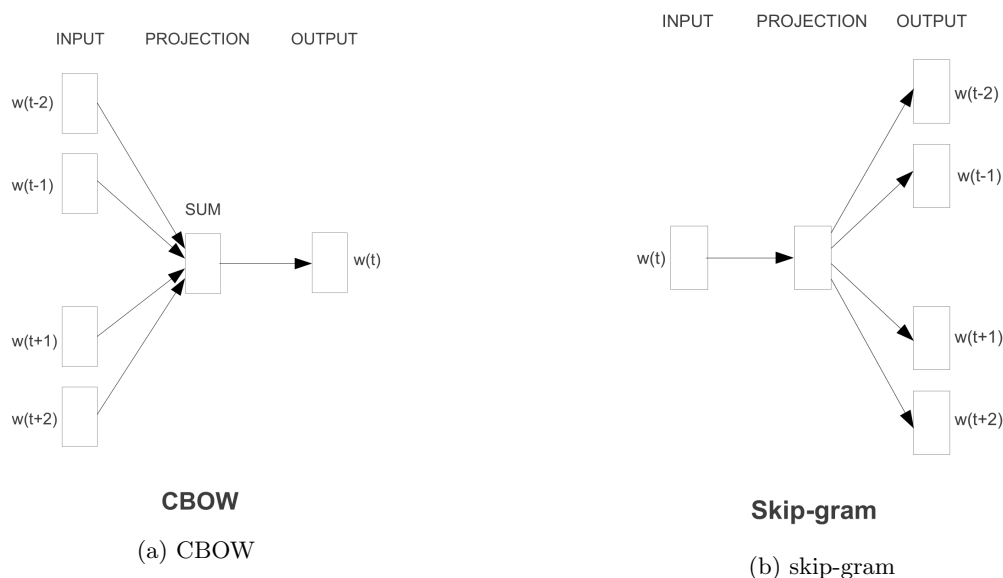


Figure 12: Architectures used in word2vec

O word2vec, proposto por (Mikolov et al., 2013) é uma ferramenta que fornece uma implementação eficiente das arquiteturas CBOW (continuous bag-of-words), Figura 12a, e skip-gram, Figura 12b, 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 exemplo clássico é a operação entre os vetores das palavras King - Man + Woman. O embedding mais próximo deste resultado é o da palavra Queen.

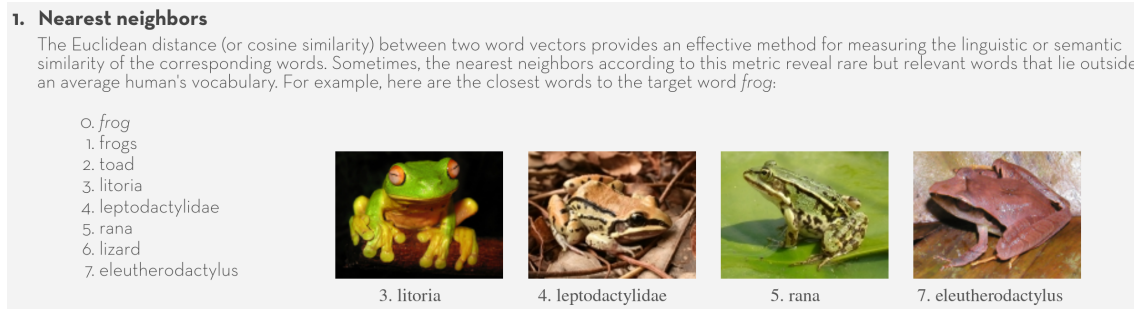


Figure 13: Embeddings vizinhos ao embedding da palavra “frog”

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 e também mostrado na Figura 13, onde é mostrado que as palavras cujos embedding mais próximos do embedding da palavra frog são: frogs, toad, litoria, leptodactylidae, rana, lizard e eleutherodactylus.

10.2 t-SNE

A técnica t-distributed Stochastic Neighbor Embedding, ou apenas t-SNE, proposta em (Maaten & Hinton, 2008) teve como autor Laurens Maaten sob orientação de um dos maiores nomes na área de machine learning, Geoffrey Hinton, tendo como revisor do artigo Yoshua Bengio, também um dos maiores nomes da atualidade na área. Esta é uma técnica de aprendizado não-supervisionado e não-linear usada como redução de dimensionalidade (geralmente usada para visualizar dados em alta dimensão). Pontos similares nessa dimensão elevada são convertidos em probabilidades conjuntas e procura-se minimizar a divergência de Kullback-Leibler (KL) entre os dados de alta dimensão e as probabilidades conjunta dos embeddings de dimensão reduzida. Geralmente trabalha-se com dimensão 2 ou 3, por questões de interpretação visual.

Para calcular as probabilidades conjuntas, primeiro centra-se uma Gaussiana (tendo como hiperparâmetro uma variável denominada perplexidade) em cada ponto na dimensão original e elevada, mede-se a densidade de todos os pontos sob a distribuição Gaussiana e por fim renormaliza-se os pontos. Isso nos dá um conjunto de probabilidades para todos os pontos, que chamaremos de P_{ij} . Em seguida, a mesma coisa é feita, porém usando uma distribuição Student t-distribution. Isso nos dá um segundo conjunto de probabilidades Q_{ij} no espaço de dimensão reduzida. Finalmente, o último passo, conforme já citado anteriormente, consiste em minimizar a divergência de KL através de uma técnica de gradiente descendente.

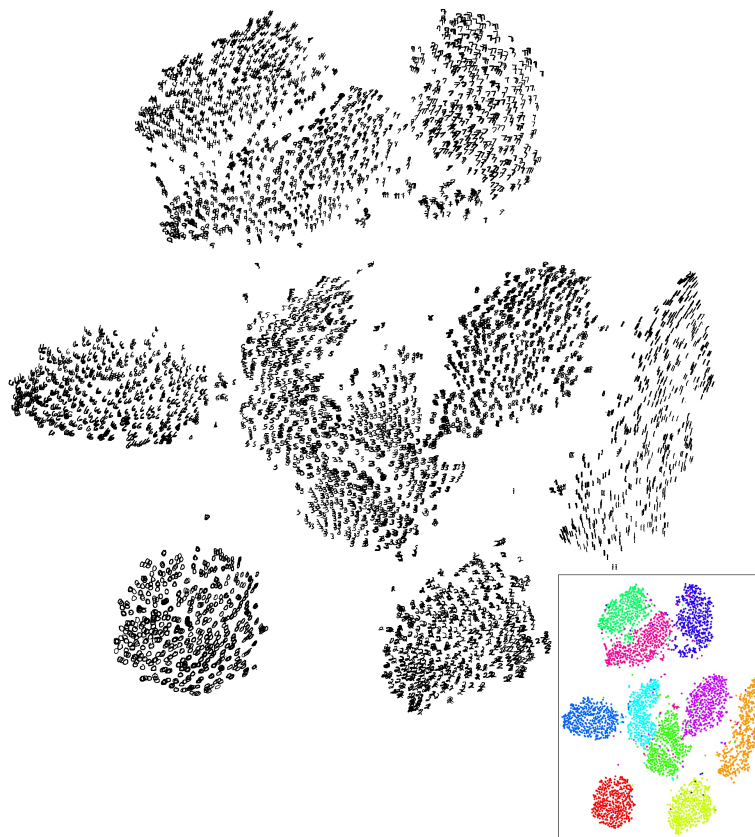


Figure 14: Exemplo da técnica t-SNE aplicada na base MNIST

Quando os dados estão em uma dimensão muito elevada, recomenda-se inicialmente reduzir a dimensão usando outra técnica (por exemplo PCA ou TruncatedSVD para uma dimensão 50), para em seguida aplicar t-SNE. Essa e outras dicas podem ser vista na própria página do autor do paper, Laurens, encontrada em <https://lvdmaaten.github.io/tsne/>. Um outro detalhe importante, é que função custo usada na implementação do t-SNE não é convexa, isso faz com que diferentes resultados sejam obtidos com diferentes inicializações.

10.3 Results discussion

Após o treinamento dado por 10 épocas, brincadeiras com palavras e o modelo treinado são feitas. Para isso utiliza-se a propriedade `model.wv` e diversos métodos.

Inicialmente procura-se a palavra mais similar a palavra “dirty”. Embora nem todas as palavras sejam sinônimas, podemos ver que são palavras relacionadas, todas indicando muito provavelmente uma avaliação negativa no dataset. Isto é, quando a palavra “dirty” aparece em uma avaliação, é bem possível que uma outra palavra semelhante, como por exemplo alguma dessas da lista, também apareça, indicando algo negativo.

Em seguida, procura-se as 6 (dada pelo parâmetro “topn”) palavras mais similares à palavra “polite”, chegando em: *courteous*, *friendly*, *cordial*, *professional*, *attentive* e *curteous*. Além das topn palavras mais similares também temos o cálculo da métrica usada, que é a similaridade de cossenos. Podemos ver que de fato temos palavras próximas.

Agora procura-se as 6 palavras mais similares à palavra “france”. Aqui, temos como retorno as seguintes palavras: *canada*, *germany*, *spain*, *hawaii*, *thailand* e *mexico*. Podemos ver que todas as palavras similares também correspondem a países, embora o valor da similaridade de cossenos tenha sido inferior ao caso anterior da palavra “polite”. Podemos talvez ir até além, tentando fazer referências entre os países, como por exemplo o fato de o Canadá apresentar o francês como um dos idiomas oficiais. Alemanha e Espanha são países que fazem fronteiras

com a França. Por fim, Havaí, Tailândia e México provavelmente aparecem na lista por serem países bastante representativos na base dados, dado que trata-se de avaliações de hotéis ao redor do globo (a Terra não é plana) e tais países são considerados turísticos.

O próximo exemplo é o de palavras similares à palavra “shocked”. Novamente temos um padrão nas resposta obtidas, todas na classe de adjetivo e com a mesma terminação, indicando sensação de surpresa.

Neste próximo exemplo usa-se três palavras com contribuição positiva e uma palavra com contribuição negativa para a medida de similaridade. As palavras com contribuição positivas são bed, sheet e pillow, e a com contribuição negativa é couch. Devido ao fato de as três palavras com contribuição positiva serem itens de cama (cama, lençol e travesseiro), temos como saída também itens de cama: cobertor, edredon, colchão, fronha e colcha, ao mesmo tempo que não temos nenhum item relacionado a palavra sofá (ex: cushion e sofá).

O próximo item mede a similaridade entre as palavras “dirty” e “smelly”. Embora sujo não seja exatamente um sinônimo de malcheiroso, temos uma medida de similaridade de cossenos relativamente alta de 0.75. Isso mostra que para o dataset em questão, essas palavras são próximas, no sentido de que é bastante plausível que ambas apareçam próximas em uma avaliação de hotel, dado que ambas são adjetivos negativos usados em uma avaliação.

Quatro palavras são usadas no próximo exemplo: bed, pillow, duvet e shower. Aqui, procura-se o item que não pertence ao conjunto. Claramente podemos ver que é a palavra chuveiro. Embora quase todo hotel tenha um chuveiro, todas outras palavras são itens de cama, o que indica que os embeddings das palavras cama, travesseiro e edredon estejam bem próximas, enquanto que o embedding da palavra chuveiro esteja mais distante das demais.

Os próximos dois itens apresentam as palavras em uma forma visual através de um gráfico 2D. Para isso trabalha-se com redução de dimensionalidade de 300 (dimensão dos embeddings) para 50 usando PCA, e em seguida reduz-se para dimensão 2 utilizando a técnica t-SNE. A função tsnescluster recebe como entrada três argumentos: modelo, uma palavra (query) e uma lista de palavras. A palavra query é mostrada em vermelho. As palavras mostradas em azul são as 10 palavras mais similares calculadas a partir da query. Por fim, as palavras da lista recebida como parâmetro da função são mostradas em verde.

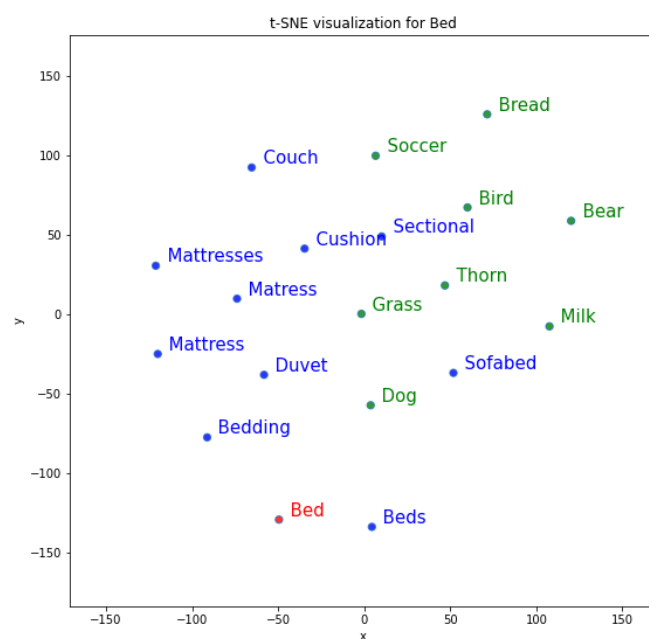


Figure 15: T-SNE 1

O primeiro gráfico é o da Figura 15. Aqui a query é a palavra “bed” e a lista de palavras contém palavras não relacionadas com a query: futebol, pássaro, cachorro, leite, etc. Podemos ver que as palavras similares (mostradas em azul) à palavra cama, encontram-se relativamente próximas da palavra “bed”. Já as palavras em verde, da lista aparentemente aleatória, encontram-se mais distantes da palavra query. Algumas exceções são visíveis, como por exemplo as palavras dog e grass, que podem ter ido parar aí devido a inicialização da técnica t-SNE, mas no geral, temos esse padrão citado anteriormente.

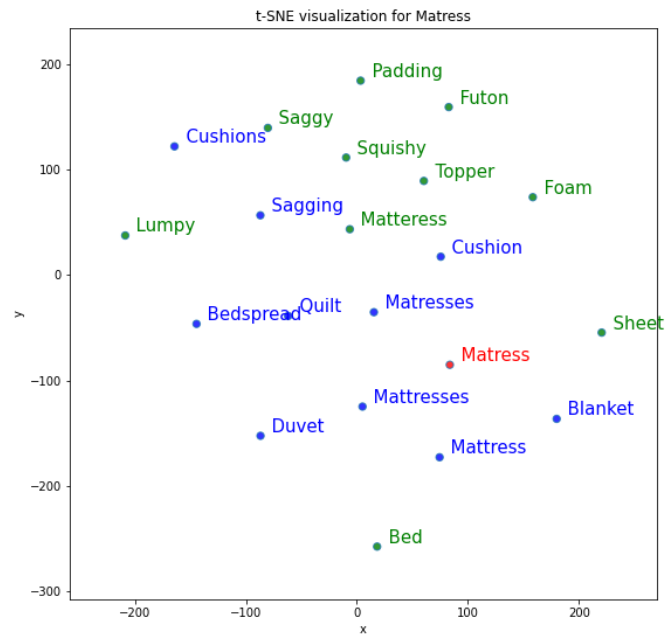


Figure 16: T-SNE 2

Por fim, o último gráfico mostrado na Figura 16 apresenta como query a palavra “matress” e recebe uma lista de 10 palavras, compreendendo as palavras entre a 10^a e 20^a mais similares a query. Dessa forma, o que temos é uma dispersão maior das palavras ao redor da query, mas ainda mantendo o padrão de as palavras azuis no geral estarem mais próximas. Uma observação importante é que query apresenta um erro de digitação (apresenta apenas um “t”, sendo escrita como “matress” ao invés de “mattress”), e curiosamente, tanto o plural correto (mattresses) quanto o plural escrito de forma equivocada (matresses) aparecem entre as palavras mais similares, indicando que essas palavras encontram-se no vocabulário do dataset e ainda que são usadas em contextos semelhantes.