ADL HW 1

B08902134, 曾揚哲

1 Data Preprocessing

I used the sample code for data processing. For both intent classification and slot tagging, the data processing procedure are the same and can be divided into following parts:

1. Collect all words that appear in training and validation set
2. Collect the labels (intent, tags) in the dataset
3. Process the labels to indices for convenience of training and testing
4. Build the embeddings by
   (a) Filter the commonly used words
   (b) Tokenize the word into integers, with padding and unknown words being 0 and 1, respectively
   (c) Calculate the GloVe embedding based on the tokenized words
5. The pre-trained embedding I used is the GloVe embedding [1].
6. Technical details: the preprocessed data are dumped onto the disk and can be loaded for future use
   (a) labels are dumped in the **json** format
   (b) common words are dumped into binary format as `vocab.pkl`
   (c) embeddings are stored as torch tensor as `embeddings.pt`

After the previously cached data, the dataset (`torch.utils.data.Dataset`) could be built on top of them. The data fed to the training procedure would be processed in the `collate_fn`, where the text (or tokens and tags) are pad to the max length specified in `args`. In intent classification task, only alphanumeric characters and spaces are retained.

2 Intent Classification

2.1 Model Structure

The baseline models I used consist of embedding layer, an optional intermediate MLP, recurrent network (RNN, LSTM, GRU) and the output MLP.
Each batch consist of N samples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$, $|\mathbf{x_i}| = l \ \forall i \in [1, N]$ where l is the designated max length to which we padded, and $y_i \in [0, 150) \ \forall i \in [1, N]$ is the intent.
For convenience, denote the batched text by $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$.
Consider the input $\mathbf{x}$ of size (N, l), we first obtain the embedding, $\mathbf{x}_{\text{embed}}$ through the embedding layer.

$$\mathbf{x}_{\text{embed}} = \text{Embeddings}(\mathbf{x})$$

While the baseline model used on kaggle does not contain the intermediate MLP, I tried the scenario with it but the performance didn't differ a lot. What this MLP does is trying to reduce the noise of the embedding, and the clean embedding is calculated as follow, the weight matrix $\mathbf{w}$ of size 300×300 (the dimension of the embedding) and the bias b are learnable.

$$\mathbf{x}'_{\text{embed}} = \mathbf{x}_{\text{embed}} \mathbf{w}^T_{300\times300} + b$$

Then, we learn the temporal info by feeding the embeddings into the recurrent network. In baseline model, I used the GRU with two layers in this module. For each layer, the reset r_t, update z_t and new gates n_t are calculated with the hidden state as follow. The hidden size is set to 256 and I applied bidirectional GRU, so the output of GRU will be of shape $(N, l, 512)$.

$$\begin{aligned}
r_t &= \sigma(W_{ir}\mathbf{x}_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}\mathbf{x}_t + b_{iz} + W_{iz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}\mathbf{x}_t + b_{in} + W_{in}h_{(t-1)} + b_{hn}) \\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{(t-1)}
\end{aligned}$$

While we have the output of shape $(N, l, 512)$, we want to obtain the temporal information in one single phase. Initially, I took the features from the last phase, $\mathbf{x}_{\text{GRU}} = \mathbf{x}^{(n)}_l$, n being the number of layers in recurrent network as the representative. While the performance can somehow pass the baseline using the last features, I also experimented on another scenarios, using the sum of all features would have better performance. That is,

$$\mathbf{x}_{\text{GRU}} = \sum_{i=0}^{l-1} \mathbf{x}^{(n)}_i$$

And $\mathbf{x}_{\text{GRU}}$ has the shape of $(N, 512)$.
Finally, the last MLP projects the features encoded with temporal information $\mathbf{x}_{\text{GRU}}$ to the dimension of output space, i.e., intents. Also, dropout is used to prevent overfitting.

$$\mathbf{x}_{\text{MLP}_{(N,150)}} = \text{RReLU}(\text{RReLU}(\mathbf{x}'_{\text{GRU}} \mathbf{w}^T_{512,512} + b)\mathbf{w}^T_{512,150} + b)$$

The prediction can be obtained by finding the index with largest probability. That is,

$$\hat{y} = \text{argmax}(\mathbf{x}_{\text{MLP}_{(N,150)}})$$

2.2 Performance

train acc	validation acc	public score
0.99420	0.93330	0.92622

2.3 Loss Function

I use cross entropy loss with l2 regularization ($\lambda = 10^{-5}$) to find the maximum likelihood setting and prevent overfit. Let $\mathbf{W}$ denote the parameters of the model, we are trying to minimize

$$E(,\mathbf{x}, y) = \frac{1}{N}\left[\sum_{n=1}^{N} -\ln\left(\frac{\exp(\hat{y}_{n,y_n})}{\sum_{c=1}^{150}\exp\hat{y}_{n,c}}\right) + \lambda\mathbf{W}^T\mathbf{W}\right]$$

2.4 Optimization

I used Adam optimizer with a learning rate of 10^{-3} and the batch size is 64.

3 Slot Tagging

3.1 Model Structure

The baseline models I used is basically the same as that I used for intent classification, consisting an embedding layer, intermediate MLP, a recurrent network, and an output MLP.
Each batch consist of N samples $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$, $|\mathbf{x_i}| = |\mathbf{y_i}| = l \; \forall i \in [1, N]$ where l is the designated max length to which we padded, and $\mathbf{y}_i[j] \in [0, 10)$, $0 \le j < l$. We are given 9 types of tags while I added one for padding to make it 10.
For convenience, denote the batched text by $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$ and the batched label by $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N)$.
Consider the input $\mathbf{x}$ of size (N, l), we first obtain the embedding, $\mathbf{x}_{\text{embed}}$ through the embedding layer.

$$\mathbf{x}_{\text{embed}} = \text{Embeddings}(\mathbf{x})$$

I tried the scenario with the intermediate MLP and the performance did improve! The MLP eliminates the noise of the embedding, and the clean embedding is calculated as follow, the weight matrix $\mathbf{w}$ of size 300×300 (the dimension of the embedding) and the bias b are learnable.

$$\mathbf{x}'_{\text{embed}} = \mathbf{x}_{\text{embed}} \mathbf{w}^T_{300\times300} + b$$

Then, we learn the temporal info by feeding the embeddings into the recurrent network. In baseline model, I used the GRU with two layers in this module. For each layer, the reset r_t, update z_t and new gates n_t are calculated with the hidden state as previously mentioned. The hidden size is set to 128 and I applied bidirectional GRU, so the output of the GRU will be of shape $(N, l, 256)$. In this task,

$$\mathbf{x}_{\text{GRU}} = \mathbf{x}^{(n)}_l$$

Finally, the last MLP projects the features encoded with temporal information $\mathbf{x}_{\text{GRU}}$ to the dimension of output space, i.e., tags. Also, dropout is adopted to prevent overfitting.

$$\mathbf{x}_{\text{MLP}_{(N,l,10)}} = \text{RReLU}(\text{RReLU}(\mathbf{x}'_{\text{GRU}} \mathbf{w}^T_{256,256} + b) \mathbf{w}^T_{256,10} + b)$$

The prediction can be obtained by finding the index with largest probability. That is,

$$\hat{y} = \text{argmax}(\mathbf{x}_{\text{MLP}_{(N,l)}})$$

3.2 Performance

train acc	validation acc	public score
0.95500	0.83800	0.78176

3.3 Loss Function

I simply used cross entropy loss to find the maximum likelihood setting. Let $\mathbf{W}$ denote the parameters of the model, we are trying to minimize

$$E(,\mathbf{x}, y) = \frac{1}{N} \sum_{n=1}^{N} \sum_{c=0}^{9} -\ln \left(\frac{\exp(\mathbf{x}_{n,y_n})}{\sum_{i=0}^{9} \exp(x_{n,i})} \right)$$

3.4 Optimization

I used Adam optimizer with a learning rate of 10^{-3} and the batch size is 64.

4 Sequence Tagging Evaluation

The result of running `classification_report(scheme=IOB2, mode='strict')` is

```
(adl) → HW1 python3 seq_eval.py
              precision    recall  f1-score   support

        date       0.76      0.77      0.76       206
  first_name       0.95      0.97      0.96       102
   last_name       0.86      0.76      0.80        78
      people       0.66      0.74      0.70       238
        time       0.81      0.87      0.84       218

   micro avg       0.77      0.81      0.79       842
   macro avg       0.81      0.82      0.81       842
weighted avg       0.78      0.81      0.79       842
```

Metrics in `seqeval`:
Denote True positive by TP, True negative by TN, False postivie by FP, False Negative by FN.

1. Precision
$$\frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{predicted positive}}$$
2. Recall
$$\frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{actually positive}}$$
3. F1-score
$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$
4. Support: The support is the number of samples of the true response that lie in that class.

Other metrics:

1. Joint Accuracy:
$$\frac{1}{N} \sum_{i=1}^{N} \{\mathbf{y}_i == \hat{\mathbf{y}}_i\}$$
A **sample** is considered correct if and only if all tokens are predicted correctly.
2. Token Accuracy
$$\frac{1}{N \sum_{i=1}^{N} |\mathbf{y}_i|} \sum_{i=1}^{N} \sum_{j=1}^{|\mathbf{y}_i|} \{\mathbf{y}_{i,j} == \hat{\mathbf{y}}_{i,j}\}$$
It is not necessary that all tokens are predicted correctly in a sample but it would count if a token is done so.

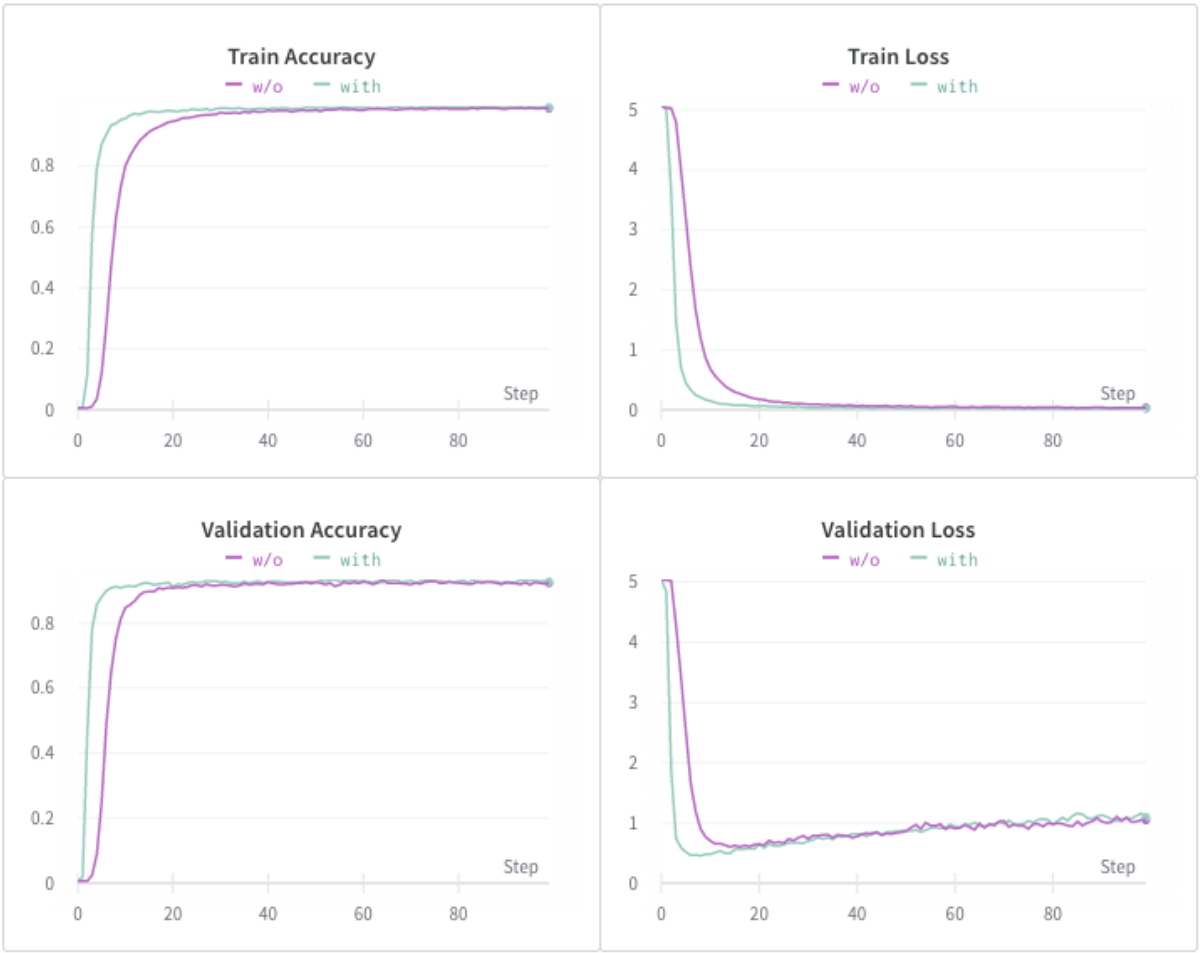
5 Comparison

5.1 Intent Classification

5.1.1 Intermediate MLP

Following the flow in section 2, the difference between whether adding intermediate MLP is not obvious. To be honest, adding it makes the performance slightly worse, but it may improve if we set the MLP deeper. The following figure presents the graph of training and validation loss and accuracy. The smaller model, i.e. the one without mid-MLP, converges faster, but their final result looks same.

	Training Accuracy	Train Loss	Validation Accuracy	Validation Loss
w/o	0.994	0.01799	0.9271	1.093
w	0.9895	0.03655	0.923	1.044



5.1.2 Different x_{GRU}

As mentioned in section 2, I take two different types of x_{GRU} for training. The first one is simply taking the features from the last index and assume it contains previous information, the other one is summing up all features, who lead to great improvement in performance. Notice that the validation accuracy greatly improved with the latter method and the loss "increases" slightly.

x_{GRU}	Training Accuracy	Train ing Loss	Validation Accuracy	Validation Loss
$\mathbf{x}_l^{(n)}$	0.9934	0.02877	0.898	2.24
$\sum_{i=0}^{l-1} \mathbf{x}_i^{(n)}$	0.9977	0.007578	0.9189	1.238

5.1.3 Different Number of Layers

I also tried with GRU with $\{4, 2, 1\}$ layers. The more layer a model consist of, the more complex it is. It is obvious that the 1-layer model converges faster than two others in terms of training loss. Moreover, we could see that the more complicated models have greater tendency to overfit if we take a peek at the graph of validation loss. Notice that these are trained with intermediate layers.

layers	Training Accuracy	Train Loss	Validation Accuracy	Validation Loss
4	0.9944	0.02039	0.9053	1.609
2	0.9895	0.03655	0.923	1.044
1	0.9938	0.01994	0.9333	0.8909

5.1.4 Different Hidden Dimension

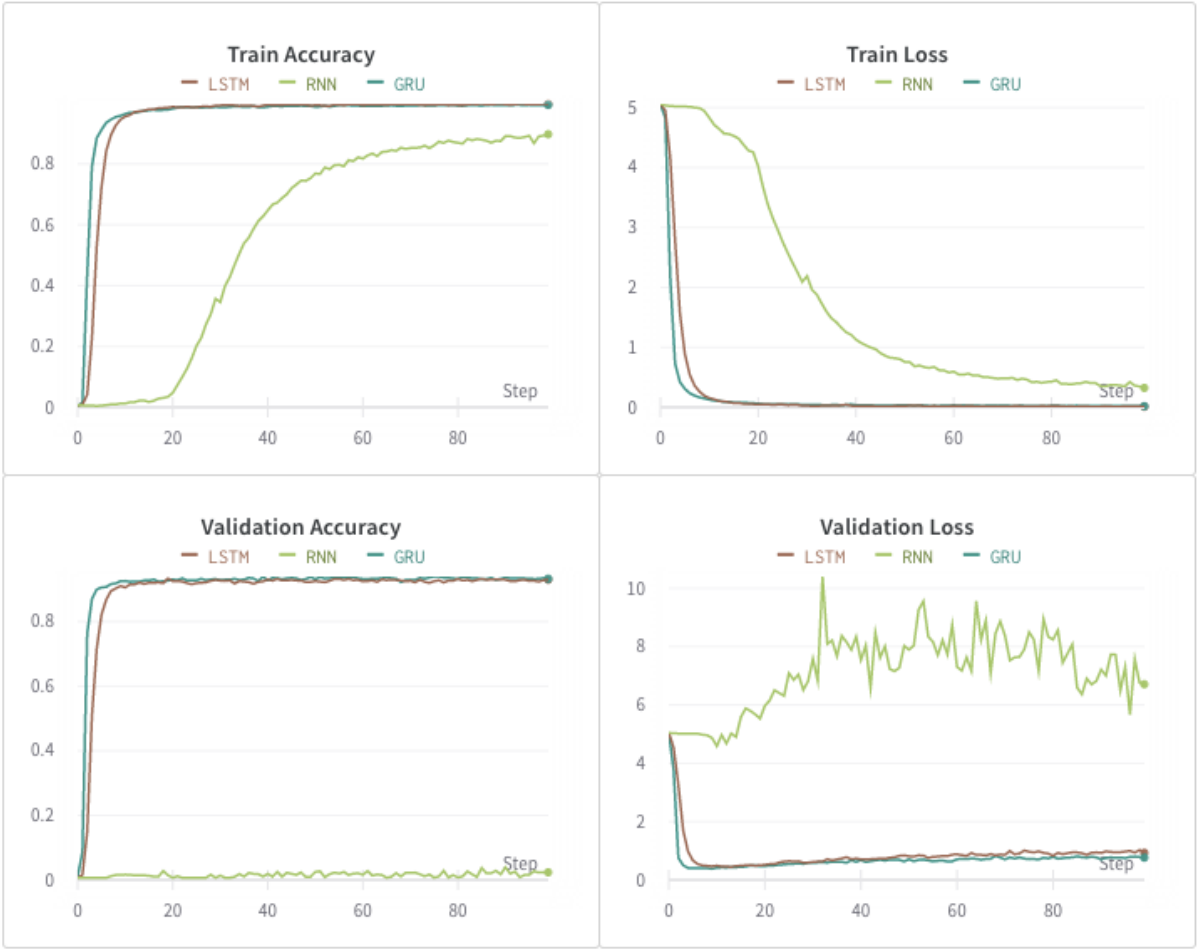
I also tried with GRU with hidden dimension of $\{512, 256, 128\}$. Surprisingly, the model with hidden dimension of 128 converges slower despite its simpler architecture, and has higher validation loss. The performance of lower dimension seems to worse probably because the smaller model cannot contain the information of the sentence considering `max_len` = 128.

dim.	Training Accuracy	Train Loss	Validation Accuracy	Validation Loss
512	0.9908	0.02957	0.9262	1.102
256	0.9948	0.01381	0.9205	1.253
128	0.9951	0.01671	0.9183	1.375

5.1.5 Different Recurrent Network

Finally, I tried to train the model with {RNN, LSTM, GRU}, respectively. The model with LSTM has pretty great performance and similar curves to those of GRU's, while GRU converges slightly faster and has resulted in lower loss on validation set. RNN without gates and further modification is indeed difficult to train and suffer gradient optimization issues. The performance on training set is somewhat acceptable, but that on validation set is terrible and often fluctuating sharply.

type	Training Accuracy	Train Loss	Validation Accuracy	Validation Loss
RNN	0.8979	0.323	0.02346	6.712
LSTM	0.9974	0.008323	0.9299	0.9271
GRU	0.9942	0.01793	0.9332	0.7642



5.1.6 Interactive

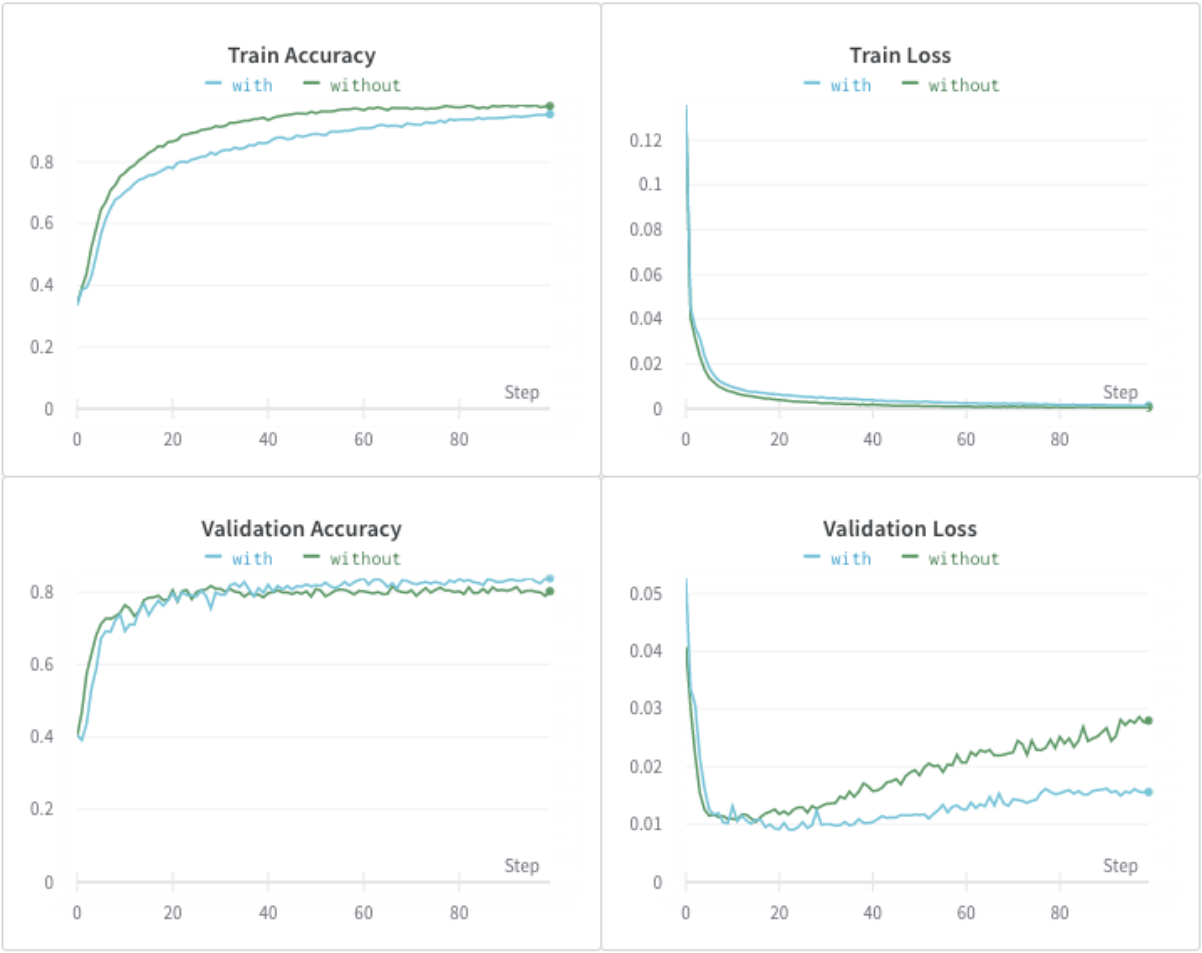
All graphs could be accessed interactively in this link.

5.2 Slot Tagging

5.2.1 Different x_{GRU}

As mentioned previously, I tried slot tagging task with different model architectures, with or without the intermediate MLP. The following graph clearly demonstrates that the one with mid-MLP improves.

	Training Accuracy	Train Loss	Validation Accuracy	Validation Loss
w/o	0.9819	0.000552	0.802	0.02796
w	0.955	0.001362	0.838	0.01562



5.2.2 Interactive

The graphs could be accessed interactively in this link.

References

[1] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014. https://nlp.stanford.edu/projects/glove/