

SPINE: SParse Interpretable Neural Embeddings Project Report

Name: Qiushi Wang
Matrikelnummer: 10033405
Date: Deep Learning 2021

- What did the paper do?

They employ a denoising k-sparse autoencoder to obtain SParse Interpretable Neural Embeddings (SPINE), a transformation of input word embeddings.

They train the autoencoder using a novel learning objective and activation function to attain interpretable and efficient representations.

They evaluate SPINE using a large scale, crowdsourced, intrusion detection test, along with a battery of downstream tasks. They note that SPINE is more interpretable and efficient than existing state-of-the-art baseline embeddings.

- How to do it?

Model:

Let $\mathcal{D} = [X_1, X_2, X_3, \dots, X_V]^T \in \mathbb{R}^{V \times d}$ be the set of input word embeddings.

Let \widetilde{X}_i be the predicted output for an input embedding $X_i \in \mathcal{D}$.

$$Z^{(X_i)} = f(X_i W_e + b_e)$$

$$\widetilde{X}_i = Z^{(X_i)} W_o + b_o$$

The loss function we want to minimize:

$$L(\mathcal{D}) = RL(\mathcal{D}) + \lambda_1 ASL(\mathcal{D}) + \lambda_2 PSL(\mathcal{D})$$

where $RL(\mathcal{D})$ is the reconstruction loss: $RL(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{X \in \mathcal{D}} \|X - \widetilde{X}\|_2^2$, where we

add a Gaussian noise on input data and use the denoised data to compute the reconstruction loss between it and the output; $ASL(\mathcal{D})$ is the average sparsity loss:

$ASL(\mathcal{D}) = \sum_{h \in \mathcal{H}} (\max(0, \rho_{h, \mathcal{D}} - \rho_{h, \mathcal{D}}^*))^2$, which penalizes any deviation of the observed

average activation value from the desired average activation value of a given hidden

unit, over a given data set; and $PSL(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{X \in \mathcal{D}} \sum_{h \in \mathcal{H}} (Z_h^{(X)} \times (1 - Z_h^{(X)}))$ obtain

an ASL value of 0 without actually having k-sparse representations.

$$\text{Use cap - ReLU}(t) = \begin{cases} 0 & \text{if } t \leq 0 \\ t & \text{if } 0 < t < 1 \\ 1 & \text{if } t \geq 1 \end{cases} \text{ as activation function.}$$

Evaluation:

Word Similarity Task: Using the WS-353 dataset which contains 353 pairs of English words. Each pair of words has been assigned similarity ratings by multiple human annotators. Using the cosine similarity between the embeddings of each pair of words, and report the Spearman's rank correlation coefficient between the human scored list and the predicted similarity list.

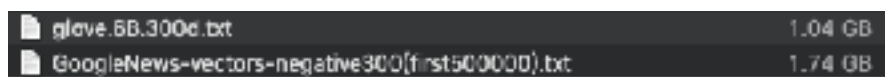
News Classification: Consider three binary news classification tasks from the 20 Newsgroups dataset². Each task involves categorizing a document according to two related categories (1) Sports: baseball vs. hockey (958/239/796) (2) Computers: IBM vs. Mac (929/239/777) (3) Religion: atheism vs. christian (870/209/717). We only use computers region here.

Noun Phrase Bracketing: We evaluate the word vectors on NP bracketing task, wherein a noun phrase of 3 words is classified as left bracketed or right bracketed. The NP bracketing dataset contains 2,227 noun phrases split into 10 folds. We append the word vectors of three words to get feature representation. For words not present in the subset of 17K words we have chosen, we use all zero vectors. We tune on the first fold and report crossvalidation accuracy on the remaining nine folds.

- What do we do?

Task 1: Reproduce results on word2vec and GloVe embeddings

As in fig.1, I downloaded the pre-trained Glove and Word2Vec embeddings, and let them be the input of the the python notebook : *Get_SPINE_Embeddings.ipynb*.

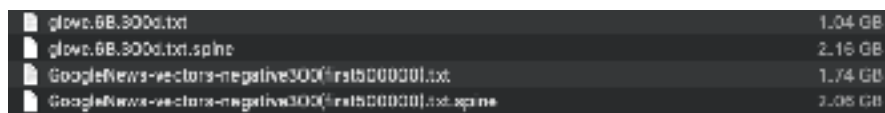


glove.6B.300d.txt	1.04 GB
GoogleNews-vectors-negative300(first500000).txt	1.74 GB

fig.1 pretrained word embeddings .txt file

As in the model, there is only two full connected layer to form the neural network. We use a noised data as input and compute the reconstruction loss between output and the denoised data. We compute the PSL loss and the ASL loss as well the total loss in training process recalling the *forward()* function.

After separate training we can get two .txt.spine file which are the SPINE-embeddings of Glove and Word2Vec embeddings.



glove.6B.300d.txt	1.04 GB
glove.6B.300d.txt.spine	2.16 GB
GoogleNews-vectors-negative300(first500000).txt	1.74 GB
GoogleNews-vectors-negative300(first500000).txt.spine	2.06 GB

fig.2 SPINE-embeddings .txt.spine file

Obtain the spine-embeddigns, we should evaluate it now. I use the original Glove and Word2Vec embeddings and their spine-embeddings as input of python notebook: *Evaluation_SPINE_Embeddings.ipynb*.

As we can see in tab.1, the SPINE-embeddings performance not obvious better than the original embeddings, according to the paper, in word similarity task, the SPINE-embeddings are supposed to have smaller Spearman's rank correlation coefficients. In noun phrase bracketing task, SPINE-embeddings of Glove have better result than the original embeddings.

	Word similarity	newsgroups(computer)	np-bracketing
Original Glove	0.6012	0.8675	0.7455
SPINE Glove	0.5064	0.8419	0.7595
Original Word2Vec	0.7000	0.8162	0.7899
SPINE Word2Vec	0.4121	0.6667	0.7353

tab.1 Evaluation of SPINE-embeddings

Task 2: Run experiments with different word embeddings

Let pre-trained Fasttext embeddings as input of python notebook: *Get_SPINE_Embeddings.ipynb*, and we get the SPINE-embeddings of Fasttext.



fig.3 pretrained word embeddings .txt file and SPINE-embeddings file

The evaluation result is shown in tab.2. In word similarity task can SPINE-embeddings not be evaluated. In my opinion, the embeddings are too sparse that there are few word embeddings are full 0, therefore the word similarity cannot be calculated.

	Word similarity	newsgroups(computer)	np-bracketing
Original Fasttext	0.7333	0.7863	0.8121
SPINE Fasttext	0	0.7692	0.6584

Task 3: Run experiments with sentence embeddings

I use the dataset of Kaggle competition "*Natural Language Processing with Disaster Tweets*" and I extract the text as sentences into file *sentence.txt*.

I removed the signs and emojis of original text and lower the first letters. As we can see in python notebook: *Get_Sentence_Embeddings.ipynb*, I compute the sentence embeddings based on Glove. Simply use the average Glove embeddings of the words appeared in a sentence as the embedding of that sentence.

In python notebook: *Get_SPINE_Embeddings.ipynb* I repeat the training process on sentence embeddings. Their input dimensions are also 300.

Task 4: Run experiments with image embeddings

I download a dataset from <https://www.tensorflow.org/datasets/catalog/overview>, and I imported the project *image_embeddings* to get the embeddings of images. In python notebook: *Get_Image_Embeddings.ipynb*.

In python notebook: *Get_SPINE_Embeddings.ipynb* I repeat the training process on sentence embeddings. Their input dimensions are now 1280.

Task 5: Evaluate sensitivity to hyperparameters

I evaluated these hyperparameters of the model and the embeddings it created: learning rate, batch size and optimizer momentum. I use Glove embeddings as reference.

As we can see in python notebook: *Evaluate_Hyperparameters.ipynb*, I adjust the hyperparameters and output the embeddings using those hyperparameters, and we use python notebook: *Evaluation_SPINE_Embeddings.ipynb* to evaluate the sensitivity.

I recorded the evaluation results in three tablets: tab.1, tab.2 and tab.3. As we can see in tab.2, our spine embeddings are sensitivity to learning rate in word similarity task, the change between the result of 0.1 learning rate and the results of other learning rate is in a interval [3.40%, 35.92%] at task word similarity, in [1.56%, 9.35%] in task newsgroups, and in [0.40%, 1.25%] in task np-bracketing.

Case 1: evaluate learning rate: batch size: 64, momentum: 0

	lr=0.15	lr=0.1	lr=0.01	lr=0.001	lr=0.0001
Word similarity	0.4817	0.5064	0.5236	0.3245	0.5370
newsgroups	0.7889	0.8419	0.7928	0.7632	0.8288
np-bracketing	0.7598	0.7595	0.7657	0.7500	0.7642

tab.2 evaluate embeddings created with different learning rates

As we can see in tab.3, our spine embeddings are not very sensitivity to batch size, the change between the result of 64 batch size and the results of other batch size is much smaller.

Case 2: evaluate batch size: learning rate: 0.1, momentum: 0

	bs=16	bs=32	bs=64	bs=128	bs=256
Word similarity	0.4198	0.4446	0.5064	0.5973	0.5861
newsgroups	0.8121	0.7851	0.8419	0.7851	0.7954
np-bracketing	0.7435	0.7608	0.7595	0.7663	0.7702

tab.3 evaluate embeddings created with different batch sizes

As we can see in tab.4, our spine embeddings are a little more sensitivity to momentum than batch size.

Case 3: evaluate momentum: learning rate: 0.1, batch_size: 64

	momentum=0.1	momentum=0.3	momentum=0.6	momentum=0.9	momentum=0
Word similarity	0.5053	0.4748	0.3963	0.4302	0.5064
newsgroups	0.7799	0.7632	0.7915	0.7992	0.8419
np-bracketing	0.7550	0.7650	0.7573	0.7600	0.7595

tab.4 evaluate embeddings created with different optimizer momentums

Task 7: Are both sparsity losses required?

Run the python notebook: *Evaluate_Without_Sparsitylosses.ipynb*. The results are the inputs of *Evaluation_SPINE_Embeddings.ipynb*. As we can see in tab.5, there are only a little difference between the result with and without one of the sparsity losses, so not the both sparsity required.

	Word similarity	Newsgroups	np-bracketing
With both loss	0.5064	0.8419	0.7595
Without psl loss	0.5396	0.7941	0.7618
Without asl loss	0.5996	0.7709	0.7483

tab.5 evaluate embeddings without one of the two sparsity losses