Report

➔ Word Embedding

There are many techniques that have been introduced prior to word embeddings like,

One-Hot Vectors -> Which is used to create a sparse vector representation of the words. Not very efficient and huge dimensions.

WordNet -> This is a dictionary-based python API which provides you with a synonym for a word in every aspect of POS [e.g., noun, adj, adverb etc.,] and also hypernyms that defines "is a" relationship [e.g., Dog – 'animal', 'organism', 'living_thing', 'mammal' etc.,]. Not all synonyms are returned and very expensive.

Distributional semantics -> This states that a word's meaning is given by the words that frequently appear close-by. Words are similar if they appear in similar contexts within a fixed-size window. We build a co-occurrence matrix with a window size, but still, it is highly sparse with huge dimensions despite of using dimensionality reduction like [SVD].

Word Embedding -> This also uses the idea of Distributional semantics but builds a dense word vector with considerable dimensions. This does bi-directional prediction.

➔ **Word2Vec** – This is a popular framework which takes a large corpus of text as input and output a vector-based representation.

Idea – The main idea involved in building the vector-based representation is that

i) Words that appear in the same context should be close i.e., their cosine similarity or dot product should be large enough.

ii) Words that don't appear in the same context should be far away i.e., their cosine similarity or dot product should be small enough.

➔ **<u>Objective Function</u>**

For a center word 'i'

O -> sum(cosine[v(i), u(i)] – sum(cosine[v(i), u(k)])

Problem 1- We do have some problems with the above objective function, it is obvious that we will have more data of the words that don't appear in the same context [i.e., Imbalanced dataset].

Solution – Negative sampling or Down sampling should solve this.

Problem 2 – Antonyms of words do often appear in the same context which makes it difficult to distinguish when building sentiment classifier.

Solution – Retraining and Remapping

- We can make this objective computationally efficient by replacing cosine similarity with dot product as the embedding values are drawn at random from same distribution.

The 2 approaches of Word2Vec are

CBOW – When we provide a context with a missing word, it predicts the missing word

Skip-gram - When given a word predicts the context of words.

➔ **<u>Multi-Sense Word Embedding</u>**

This raise two different representations of word senses

i) Polysemy - A word or a phrase with multiple meanings.
ii) Homonym – A word with same spelling or same pronunciation but has different meaning.

This can be achieved using **Word sense disambiguation (WSD)** wherein we input the context and try to predict the sense from fixed inventory of each word using baseline, supervised or unsupervised method.

➔ **<u>Multi-Lingual Embedding</u>**

This is similar to Word2Vec embedding and in addition to that it also includes words of multiple languages.

It has 2 main objectives

i) Mono-lingual
ii) Cross-lingual
iii) This can be achieved using **Mono-lingual embedding with dictionary, linear transformation** and **Bi-lingual skip-gram.**

<u>Tuning Hyperparameters of MLP Encoder</u>

<u>Hidden layers</u> – This parameter is used to decide the number of hidden layers.
<u>Solver or Optimizer</u> – Several optimizers like 'Adam', 'sgd which uses [online learning]', lbfgs [mini-batch] etc.,
<u>Regularization parameter</u> – This is used to reduce variance of the data without increasing bias. This is also called tuning parameter.
<u>Batch size</u> – The number of datapoints to be used for each forward pass before going through backward pass and adjusting weights.
<u>Learning rate</u> – We can adjust how the learning rate should vary implementing '**adaptive learning'** rate and invscaling for '**power scheduling'**.
<u>Maximum iterations or epochs</u> – Number of iterations to go through the entire data.
<u>Early stopping</u> – This is used to avoid overfitting problem by stopping the training if the loss increases above certain tolerance level
<u>Shuffle</u> – Shuffling the dataset after each epoch or for each batch.

I have tried implementing
Exponential learning rate -> Decays the learning rate of each parameter group by rate of gamma for each epoch. This increased the MSE.
Multi-step learning rate -> Decays the learning rate of each parameter group by gamma once the number of epochs reach one of the milestones. This reduced the MSE significantly.
Step learning rate -> Decays the learning rate of each parameter group by gamma every step size epochs.
Lambda learning rate -> Sets the learning rate of each parameter group to the initial learning rate times a given function.

we can also switch learning rates after each epoch by calling them one after the other.

Tuning Hyperparameters of CNN

Stride – This parameter of type tuple or list is used to define the rate at which the kernel or filter to pass over rows
padding – Adding zeros on top and bottom of the input shape to make all the input data used same number of times.
Filter or kernel size – Size of the kernel.
Use bias – Specify the layer to use bias vector.
Activation – Provide the activation function to be used.
Max-t – In max-over-time-pooling instead of obtaining top value in convolution output obtain top t values using this parameter.