Report

1. Hyper-parameters explored w/ a description of what each parameter controls and how varying each is likely to affect the training. These are the parameters I suggest to tune: max num steps, batch size, skip window, num skips, num sampled

    a. **Skip window**: This tell us the number of words to consider as neighbours to the left and right of the target word. i.e tells the exact number of words to the right of the target word and to the left of the target word.
    Smaller window size gives results that are more syntactic in nature. Larger window gives results that are more semantic. Smaller window sizes (2-15) lead to embeddings where high similarity scores between two embeddings indicates that the words are interchangeable. Larger window sizes (15-50) lead to embeddings where similarity is more indicative of relatedness of the words. A larger window size takes more time for training.

    b. **Num sampled**: it is the number of negative samples. Given the huge size of the word vocabulary, the training will include a large number of weights and all of them will get updated even if slightly by each and every training sample. Negative sampling helps in tackling this scenario by having each training sample modify a small percentage of the weights, rather than all of them. With negative sampling, we randomly select a small number of “negative” words to update the weights for. We will still update the weights for our “positive” words.
    Higher number of negative samples work well for smaller datasets and less number of negative samples work well for larger datasets. According to the paper the smaller datasets typically need 5-20 negative samples and larger documents need < 5 negative samples.

    c. **Num_skips**: How many times to reuse an input to generate a label. This value defines the number of (input,label) tuples each word generates. So num_skips restrict the number of context words we would use as output words. The code randomly pick up num_skip context words to construct training pairs with target. The larger the value of num_skips, more will be the number of context words for a given target word, which may result in underfitting. Also, using very low num_skips may restrict only one context word with the target word.

    d. **Embedding size**: This is the dimension of the word embedding. Dimension size usually ranges between 100-300. However, if you choose dimension size bigger than 300, then it has a diminishing effect on your model.

    e. **Max_num_steps**: This tells us the maximum number of training cycles. In each step we cycle through all the training samples. Increasing the number of training steps helps in training the model better and reducing the loss.

However, the model might stop improving after a certain number of training cycles or the improvement might be extremely small. The max_num_steps should not be too small to allow sufficient training.

f. **Batch_size**: The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Batch can be thought as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model. The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.

2. Results on the analogy task for five different configurations of hyper- parameters, along with the best configuration (on the word pairs dev.txt). Explain these results. Are there any trends you observe? For example, what does increasing dimensions of vectors do? You should say what you learned from exploring these configurations. You are free to consult any resources to support your arguments.

Hyperparameter configuration					**Overall Accuracy:**	
batch_size	**skip_window**	**num_skips**	**num_sampled**	**max_num_steps**	**cross entropy**	**nce loss**
128	**4**	**8**	**64**	**200001**	**34**	**34.2**
100	4	5	64	200001	33.6	32.5
120	5	10	64	300001	33.9	31
128	1	2	32	100001	31.7	31.8
256	4	8	64	200001	29.6	32.5
512	4	8	32	100001	30.1	33

Hyperparameter configuration						**Accuracy of Least Illustrative Guesses**	
batch_size	**embedding_size**	**skip_window**	**num_skips**	**num_sampled**	**max_num_steps**	**cross entropy**	**nce loss**
128	**128**	**4**	**8**	**64**	**200001**	**34.1**	33
100	128	4	5	64	200001	31.3	**34.6**
120	128	5	10	64	300001	33.3	31
128	128	1	2	32	100001	29.3	30.2
256	128	4	8	64	200001	29.4	30.5
512	128	4	8	32	100001	28.2	31.9

Hyperparameter configuration						Accuracy of Most Illustrative Guesses:	
batch_size	embedding_size	skip_window	num_skips	num_sampled	max_num_steps	cross entropy	nce loss
128	**128**	**4**	**8**	**64**	**200001**	33.9	**35.4**
100	128	4	5	64	200001	36	30.4
120	128	5	10	64	300001	**34.6**	31
128	128	1	2	32	100001	34.1	33.5
256	128	4	8	64	200001	29.8	34.5
512	128	4	8	32	100001	31.9	34

Below are the trends for my experiment:

a. Both cross entropy and nce had best models for skip_window = 4. Accuracy decreased on both increasing and decreasing the skip window size.
b. Given the dataset was extremely small, I was getting best results for num sampled = 64. Tried decreasing it, but was getting lower accuracy.
c. I was getting the best accuracy for both cross entropy and nce models for num_skips = 8. Accuracy decreased on both increasing and decreasing the num_skips.
d. The best accuracy for both cross entropy and nce models were observed for embedding_size = 128. Didn't try modifying this parameter.
e. For max_num_steps, I observed that loss was reducing till step 195000, after that it did not have that much impact. So I chose max_num_steps = 200001 and both cross entropy and nce models were giving peak accuracy for this.
f. When modifying batch size I observed that, for cross entropy the model was giving best results for batch size 128. But, on reducing the batch size or on increasing the batch size the accuracy decreased. The accuracy decreases more on increasing the batch size. In case of NCE model, the accuracy was highest for batch size = 128, but it decreased more on lowering the batch size as compared to increasing the batch size.

3. Top 20 similar words according to your NCE and cross entropy model for the words:{first, american, would}. Please report these in a table. And discuss what kinds of similarities do you notice?

For Cross entropy model (format:- similarity index :: context word)
We can observe that similarities are in the form these words will get used in a sentence. Usually, the context words will follow or precede the target words in a meaningful sentence.

Sno	first	american	would
1	first	described	notes

2	aligns	quieter	pellicle
3	ranges	urraca	harsher
4	thant	enticed	miffed
5	trimotor	singularly	vogtsbauernh
6	greening	ranaldo	aana
7	petrucciani	imsa	propagandized
8	chez	explicitely	expatriated
9	pioneering	bfi	stochansky
10	baileyana	brims	succinyl
11	barrios	redistricting	lecky
12	pvps	belinda	faither
13	verran	olimpiade	agrarianism
14	koji	vitosha	icon
15	alere	handcuffs	chokepoint
16	aardwolves	emilia	lubieniecki
17	prop	gainesville	socialization
18	falconer	steinh	lieserl
19	mariculture	mayhem	pads
20	squirt	chatterjee	aga

For NCE model (format:- similarity index :: context word)
We can observe that similarities are in the form these words will get used in a sentence. Usually, the context words will follow or precede the target words in a meaningful sentence.

Sno	first	american	would
1	first	described	notes
2	of	carnot	thousandth
3	in	fireplaces	spirally
4	sans	robocup	catchy
5	revolution	coercive	warmbloodedness
6	radicals	prostitute	cimento
7	term	brims	galaxian
8	culottes	evaporative	rugians
9	used	python	casbah
10	including	dulling	infirmary

11	french	acrostic	nikko
12	english	bronte	yah
13	against	cru	thinkpad
14	is	vulgare	matriarchate
15	working	anorexic	bosworth
16	still	glow	nabataean
17	the	wingers	boulder
18	diggers	scheduling	yo
19	and	spherical	selinum
20	early	dramas	cryptovolans

Also, we observe that the most similar word for both models are the same.

4. A summary of the justification behind the NCE method loss. No more than a page. This should be a summary of section 3 (3.1 specifically) in the NCE paper. Explain how the proposed loss function models the probability distribution of a word in its context.

   The center thought of Noise Contrastive Estimation (NCE) is to change over a multiclass classification problem into one of binary classifications by means of logistic regression, while as yet holding the nature of word vectors learned. With NCE, word vectors are never again learned by endeavoring to anticipate the setting words from the objective word. Rather we learn word vectors by figuring out how to recognize genuine sets of (target, context) words from corrupted (target, arbitrary word from vocabulary) sets. The thought is that if a model can recognize real matches of target and context words from irregular noise, at that point great word vectors would be found out. In particular, for every positive example (ie, target/context pair) we present the model with k negative examples drawn from a noise distribution. For little to average size training datasets, an incentive for k somewhere in the range of 5 and 20 was prescribed, while for enormous datasets a smaller estimation of k somewhere in the range of 2 and 5 gets the job done. NCE model just has a solitary output node, which predicts whether the pair was simply arbitrary noise or really a substantial target/context pair. Noise Contrastive Estimation is a method for learning a data distribution by looking at it against a noise distribution, which we characterize. This enables us to give an unsupervised problem as supervised logistic regression problem. It's usage is like Negative Sampling, which is an estimation component that was imagined to decrease the computational expense of normalizing system outputs by adding over the whole vocabulary. The essential distinction in execution among NCE and Negative Sampling is that in NCE, the likelihood that an example originated from the noise circulation is unequivocally represented and the problem is cast as a formal estimate of the log-odds ratio that a particular sample came from the genuine data distribution rather than the noise distribution.
   The reason why NCE loss will work is because NCE approximates maximum

likelihood estimation (MLE) when the ratio of noise to real data k increases. For each data (x,y), y is the labeled class from the data, NCE loss samples k classes from noise distributions. Then we calculate a special version of the digits for each of the classes using the equation

$$\frac{\partial}{\partial\theta}J^{h,w}(\theta) = (1-\sigma\left(\Delta s_\theta(w,h)\right))\frac{\partial}{\partial\theta}\log P_\theta^h(w) - \sum_{i=1}^{k}\left[\sigma\left(\Delta s_\theta(x_i,h)\right)\frac{\partial}{\partial\theta}\log P_\theta^h(x_i)\right]$$

Where $Pn(w)$ is the noise distribution. With the digits for each classes calculated, the digits can be used to compute softmax loss for binary classification for each of the classes, and add these losses together as the final NCE loss.

The total over k noise samples rather than a total over the whole vocabulary, making the NCE training time linear in the number of noise samples and independent of the vocabulary size. As we increment the quantity of noise samples k, this estimates approaches the likelihood gradient of the normalized model, enabling us to trade off calculation cost against the accuracy.