# 1 Analogy Task

The answer for this problem is divided into the following 3 parts:

A. Results of Analogy test runs that use Word2Vec and GloVe pre-trained embeddings.

B. Comment on embeddings of antonyms.

C. Results of Analogy test runs that use custom Test questions outside Mikolov's dataset.

**Part A**

**Results**

Here are accuracy results of various analogy tasks using Word2Vec and GloVe embeddings (of different dimensions). See program `analogy.py` for more details.

| Embeddings → / ↓ Task Name | Word2Vec | GloVe 50d | GloVe 100d | GloVe 200d | GloVe 300d |
|---|---|---|---|---|---|
| capital-common-countries | 81.82 | 80.03 | 94.07 | 95.25 | **95.45** |
| capital-world | 77.8 | 68.76 | 88.41 | 94.2 | **95.97** |
| currency | **31.17** | 8.7 | 14.54 | 16.74 | 16.16 |
| city-in-state | **69.03** | 15.68 | 30.15 | 50.7 | 60.43 |
| familly | 84.38 | 70.15 | 80.03 | 83.6 | **87.74** |
| gram1-adjective-to-adverb | **27.92** | 13.61 | 23.68 | 24.90 | 22.47 |
| gram2-opposite | **39.77** | 9.23 | 20.07 | 23.64 | 26.84 |
| gram3-comparative | **89.26** | 53.73 | 79.05 | 85.66 | 87.23 |
| gram4-superlative | **85.47** | 28.25 | 54.99 | 67.55 | 70.86 |
| gram5-present-participle | **79.07** | 41.47 | 68.84 | 67.51 | 69.5 |
| gram6-nationality-adjective | 90.18 | 86.5 | 88.55 | 92.43 | **92.55** |
| gram7-past-tense | **66.34** | 35.89 | 53.39 | 58.58 | 59.67 |
| gram8-plural | **90.99** | 59 | 71.62 | 77.17 | 77.85 |
| gram9-plural verbs | **68.96** | 38.16 | 58.39 | 59.4 | 58.5 |
| *overall accuracy* | ***70.15*** | ***43.51*** | ***58.98*** | ***64.09*** | ***65.80*** |

**Observations**

- As seen in the table above, the accuracy for each individual task and overall gets better as we increase then number of dimensions from GloVe 50d all the way to GloVe 300d and Word2Vec (which is also 300d).
- The number of dimensions are low enough that there is still a possibility of increasing the dimensions to increase the accuracy. However, after a point if you continue increasing the dimensions the accuracy will actually go down since the vectors would become sparse.

- (Hypothesis) GloVe does better on the capitals of the country's (capital-common-countries, capital-world) owing to plethora of Wikipedia articles on different countries that are structured similarly.
- (Hypothesis) Google News captures currencies better since there is a whole section of news articles dedicated to finance where news related to currency is very common. Hence, Word2Vec does better on currency.

**Algorithm**

Here is the algorithm I followed to obtain these results:

Step 1: Load the word embeddings using `gensim`.

Step 2: Preprocess Analogy question and answer (strip out whitespaces, switch to lowercase if using GloVe).

Step 3: Find top 10 possible answers to each question using `gensim`.

Step 4: Compute cosine similarity between the question and possible answers.

Step 5: Use results from Step 4 to find the best guess out of the 10 possible answers.

Step 6: Compare the best guess with the provided answer. If matches, count as correct answer. If not, count as an incorrect answer.

Step 7: Repeat 3-6 for each question. Compute accuracy per task as:

$$total\_correct\_answers \,/\, total\_questions.$$

**Note:** I did not find any missing words in either Word2Vec or GloVe.

**Part B**

In order to examine the behavior, I created a program (`antonyms.py`) that displays top 10 similar words given a pre-trained embedding (GloVe or Word2Vec) using cosine similarity.

**Results**

```
Enter a word to see top 10 similar words (may include antonyms) or hit CTRL+C to quit: good
['great', 'bad', 'terrific', 'decent', 'nice', 'excellent', 'fantastic', 'better', 'solid',
'lousy']
Enter a word to see top 10 similar words (may include antonyms) or hit CTRL+C to quit: high
['low', 'High', 'higher', 'highest', 'ahigh', 'lower', 'Low', 'HIGH', 'lowest', 'elevated']
Enter a word to see top 10 similar words (may include antonyms) or hit CTRL+C to quit: enter
['entering', 'entered', 'reenter', 'enters', 'entry', 'Entering', 'participate', 'leave', 'join',
'register']
Enter a word to see top 10 similar words (may include antonyms) or hit CTRL+C to quit: increase
```

```
['decrease', 'increases', 'increased', 'reduction', 'increasing', 'decreases', 'rise',
'decreasing', 'decline', 'increasein']
```

**Observation**

- I noticed that, given a word, antonyms of that word typically fall in the top 10 most similar words. This is because antonyms and synonyms have similar word embeddings.
- The reason why antonyms and synonyms have similar word embeddings is because they are frequently used in very similar contexts.
- For example: In the fragments, "temperature increases", "temperature decreases", increases and decreases have same context and embedding algorithms take into account the context when building a word vector for a particular word.
- It is thus difficult for such embeddings to distinguish between antonyms and synonyms.

**Part C**

I have uploaded my test questions (4 of each type) as part of A2.tar.gz in a file called `word-test.v2.txt`.

**Results**

| Embeddings → ↓ Task Name | Word2Vec | GloVe 50d | GloVe 100d | GloVe 200d | GloVe 300d |
|---|---|---|---|---|---|
| languages-of-the-world | **100** | 75 | 75 | 75 | **100** |
| products | **100** | 50 | 50 | 100 | **100** |

**Observation**

- As seen in the table above, the accuracy for each individual task and overall gets better as we increase then number of dimensions from GloVe 50d all the way to GloVe 300d and Word2Vec (which is also 300d).

# 2 Sentiment Classification

The answer to this problem is divided into the following 6 parts:

A. Specification of the Feed-forward Neural Network classifier built using TensorFlow.

B. Comments on the performance of different embeddings.

C. Comments on the performance of one-hot representation.

D. Comments on the performance of Neural Network classifier vs Log-Linear classifier.

E. BONUS! Observations on a variety of parameter variation exercises.

F. Sentiment breakdown for aPhone and Adenoid users in the best model.

**Part A**

The implementation of the Feed-forward Neural Network (`sentiment_nn_2017.py`) uses a computation graph that is given by the following equation:

$$h_1 = \tanh\,(x^T\, W_1 + b_1)^T$$

$$h_2 = \tanh\,(h_1^T\, W_2 + b_2)^T$$

$$p = softmax(h_2)$$

$$\hat{y} = \arg\max p_i$$

As shown in the above equation, I have used $\tanh$ as the activation function.

**Part B**

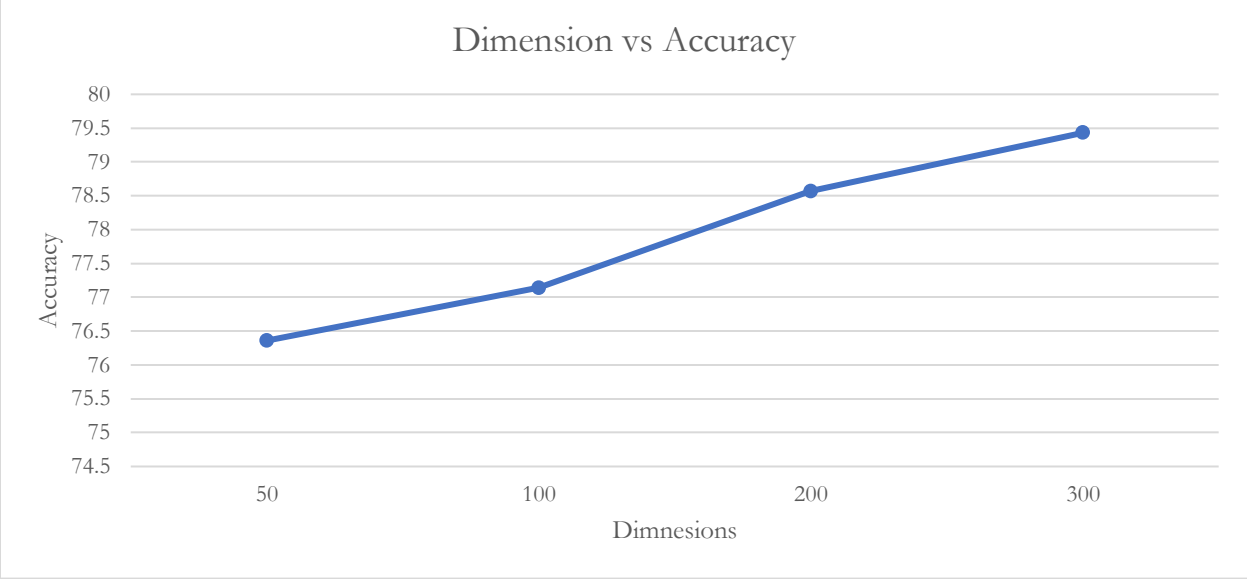
I ran the above implementation on the provided embeddings (of different dimensions and sources) by keeping the following parameters consistent:

Number of Hidden Layers: 1; Hidden Layer size: 100; Mini batch Size: 100; Epoch size: 10

**Results**

| GloVe 50d | GloVe 100d | GloVe 200d | GloVe 300d | Twitter 25d | Twitter 50d | Twitter 100d | Twitter 200d | Word2Vec 300d |
|---|---|---|---|---|---|---|---|---|
| 76.93 | 77.14 | 78.57 | 79.43 | 63.9 | 64.26 | 64.15 | 64.26 | 79.34 |

**Observations**




- As seen in the table and chart above, models that are higher in dimension i.e. GloVe 300d and Twitter 200d do better than their lower dimension counterparts.
- Overall, GloVe 300d and Word2Vec 300d yield the best performance for my implementation.
- The model performed well with embeddings that source data from Wikipedia and Google News, but it didn't do as well with Twitter as the data source for the pre-trained embeddings (even for the same dimensions – GloVe 100d vs. Twitter 100d).

**Part C**

I ran the above implementation on the one-hot representation by keeping the following parameters consistent:

Number of Hidden Layers: 1; Hidden Layer size: 100; Mini batch Size: 100; Epoch size: 10

**Results**

| One-hot | GloVe 300d | Word2Vec 300d |
|---|---|---|
| 82.52 | 79.43 | 79.34 |

**Observations**

- As seen in the table above, One-hot representation performs better than the best-performing pre-trained word embeddings.
- This is because One-hot takes into account more dimensions than GloVe or Word2Vec. Every word in the vocabulary is a dimension in One-hot representation.

- Computationally One-hot is extremely heavy and slow as the number of dimensions in One-hot representation are orders of magnitude higher than the pre-trained word embeddings.

**Part D**

I ran the Log-Linear and Feed-forward Neural Network implementation with the same mini-batch size of 100 and epoch size of 10.

**Results**

| | Log Linear Model | Neural Network Model |
|---|---|---|
| One-hot | 68.22 | 82.52 |
| Twitter 25d | 63.06 | 63.9 |
| Twitter 50d | 63.33 | 64.26 |
| Twitter 100d | 63.31 | 64.15 |
| Twitter 200d | 63.25 | 64.26 |
| GloVe 50d | 74.27 | 76.93 |
| GloVe 100d | 76.17 | 77.14 |
| GloVe 200d | 77.76 | 78.57 |
| GloVe 300d | 78.49 | 79.43 |
| Word2Vec | 76.87 | 79.34 |

**Observations**

- Performance of neural network is better than log linear model since a non-linearity term is added by neural network by means of an activation function.

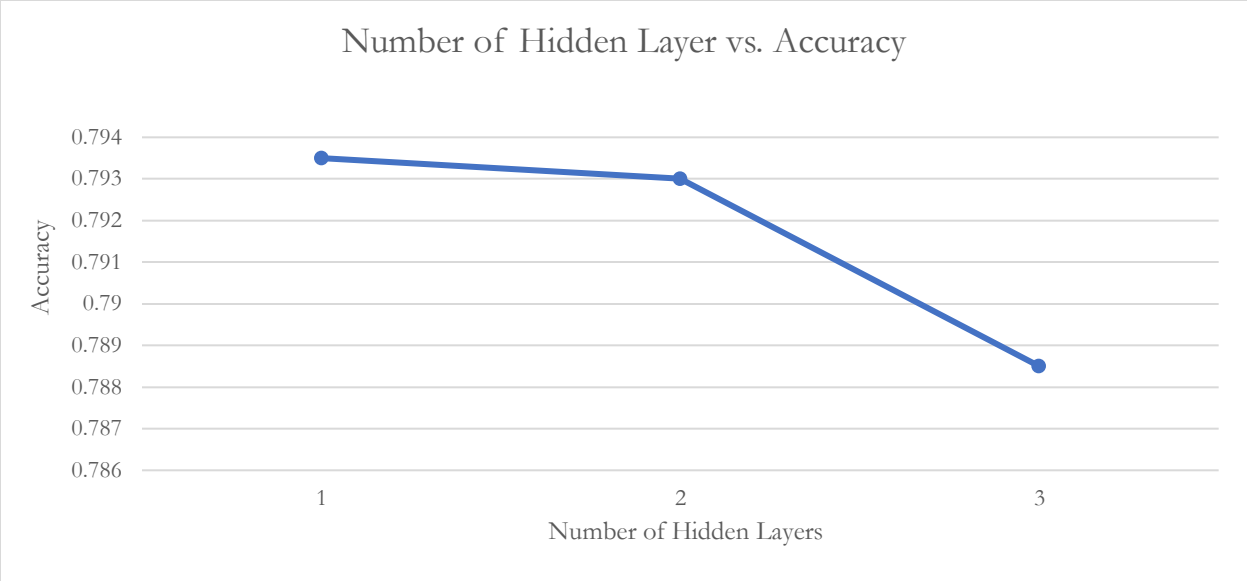
**Part E – BONUS!**

I did a variety of experiments with the number of hidden layers, number of epochs, etc. I'm presenting my findings here in this section. In each of the experiments, aside from the parameters being tested, the following are used as constants:

Number of Hidden Layers: 1; Hidden Layer size: 100; Mini batch Size: 100; Non-linearity: $tanh$

**Number of Hidden Layers**

For this experiment, I modified my implementation to add more hidden layers and examined the result. To find this implementation look for `setiment_bonus_nn_2017.py` in A2.tar.gz.

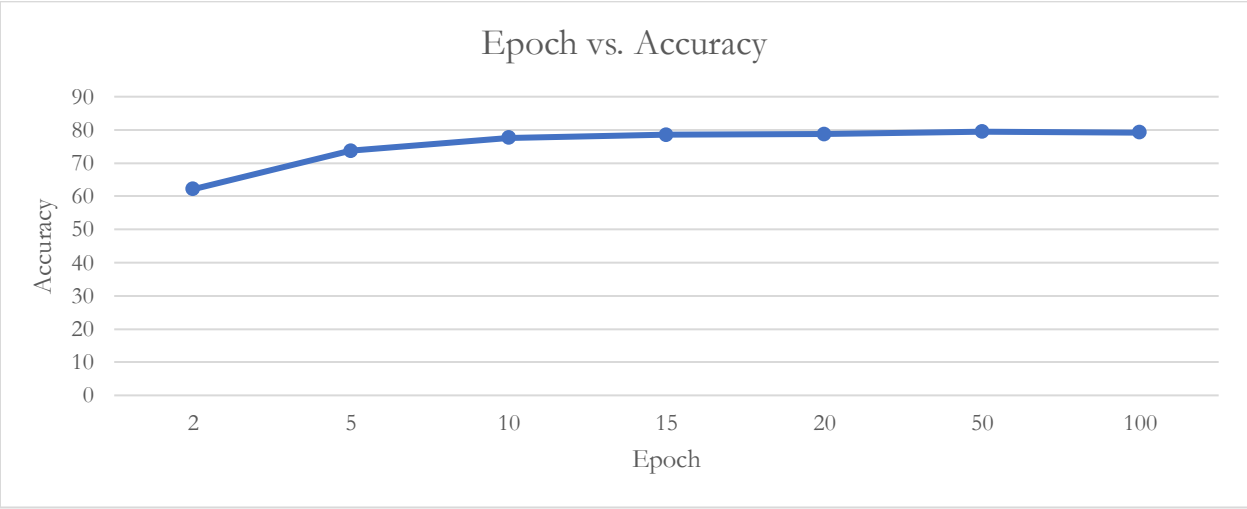



As seen in the graph above, increasing the number of hidden layers accuracy drops. Following are the equations I have used:

$$h_1 = \tanh\,(x^T\, W_1 + b_1)^T$$
$$h_2 = \tanh\,(h_1^T\, W_2 + b_2)^T$$
$$v = \tanh\,(h_2^T\, W_3 + b_3)^T$$
$$p = softmax(v)$$
$$\hat{y} = \arg\max p_i$$

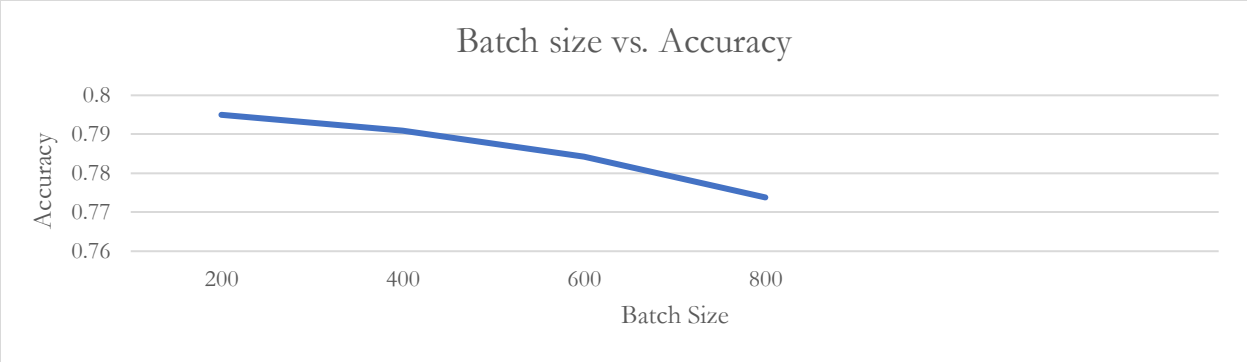
**Epoch Size**

For this experiment, I kept changing the epoch size and observed that initially accuracy increases with increase in epoch and at some point, it becomes constant.



**Batch Size**

For this experiment, I kept changing mini batch size and observed that accuracy decreases with increase in batch size.




**Activation Function**

For this experiment, I tried using different activation function. Here are my findings:

| Activation Function | Accuracy |
|---|---|
| **relu** | 78.54 |
| **tanh** | 79.41 |
| **sigmoid** | 78.74 |
| **elu** | 79.2 |

**Part F**

Based on my observations, one-hot representation has the best accuracy followed by GloVe 300d and Word2Vec 300d. Following are the results with accuracy, positive Adenoid tweets and Positive aPhone tweets:

| | Accuracy | Positive Adenoid Tweets | Positive aPhone Tweets |
|---|---|---|---|
| **One-hot representation** | 82.52 | 38.78 | 34.01 |
| **GloVe 300d** | 79.17 | 37.8 | 34.05 |
| **Word2Vec 300d** | 79.39 | 36.04 | 32.95 |

Following are the parameters used by the Feed-forward Neural Network that provides the above results: Number of Hidden Layers: 1; Size of Hidden Layer: 100; Epoch Size: 10; Mini batch Size: 100 Non-linearity: $tanh$

**References**

[1] Goldberg, Y. (2016). A primer on neural network models for natural language processing. Journal of Artificial Intelligence Research, 57, 345-420.

[2] Gensim: topic modelling for humans. (n.d.)., from https://radimrehurek.com/gensim/