Comparison of Statistical and Neural Language Modelling

1. **Dataset**
   - **1.1. Brown Corpus:** Contains sentences is an electronic collection of text samples of American English, the first major structured corpus of varied genres. This corpus first set the bar for the scientific study of the frequency and distribution of word categories in everyday language use.

```
32
33    It urged that the next Legislature "provide enabling funds
34    and re-set the effective date so that an orderly implementation of
35    the law may be effected".   The grand jury took a swipe at the
36    State Welfare Department's handling of federal funds granted for
37    child welfare services in foster homes.   "This is one of the
38    major items in the Fulton County general assistance program", the
39    jury said, but the State Welfare Department "has seen fit to distribute
40    these funds through the welfare departments of all the counties
41    in the state with the exception of Fulton County, which receives
42    none of this money.   The jurors said they realize "a proportionate
43    distribution of these funds might disable this program in our less
44    populous counties".   Nevertheless, "we feel that in the
```

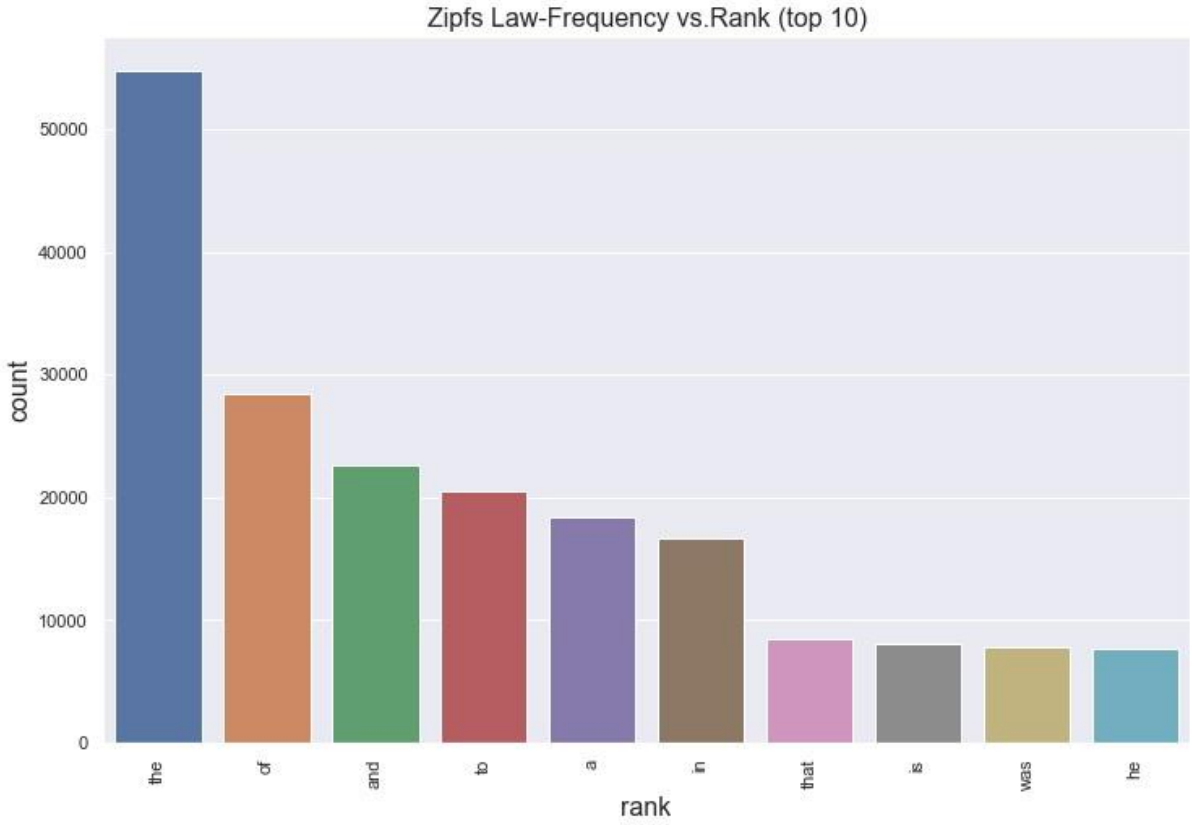
The corpus required some pre-processing, like removal of lines beginning with '#' which are just the sub-text headings, also there were blank lines which were of no use.

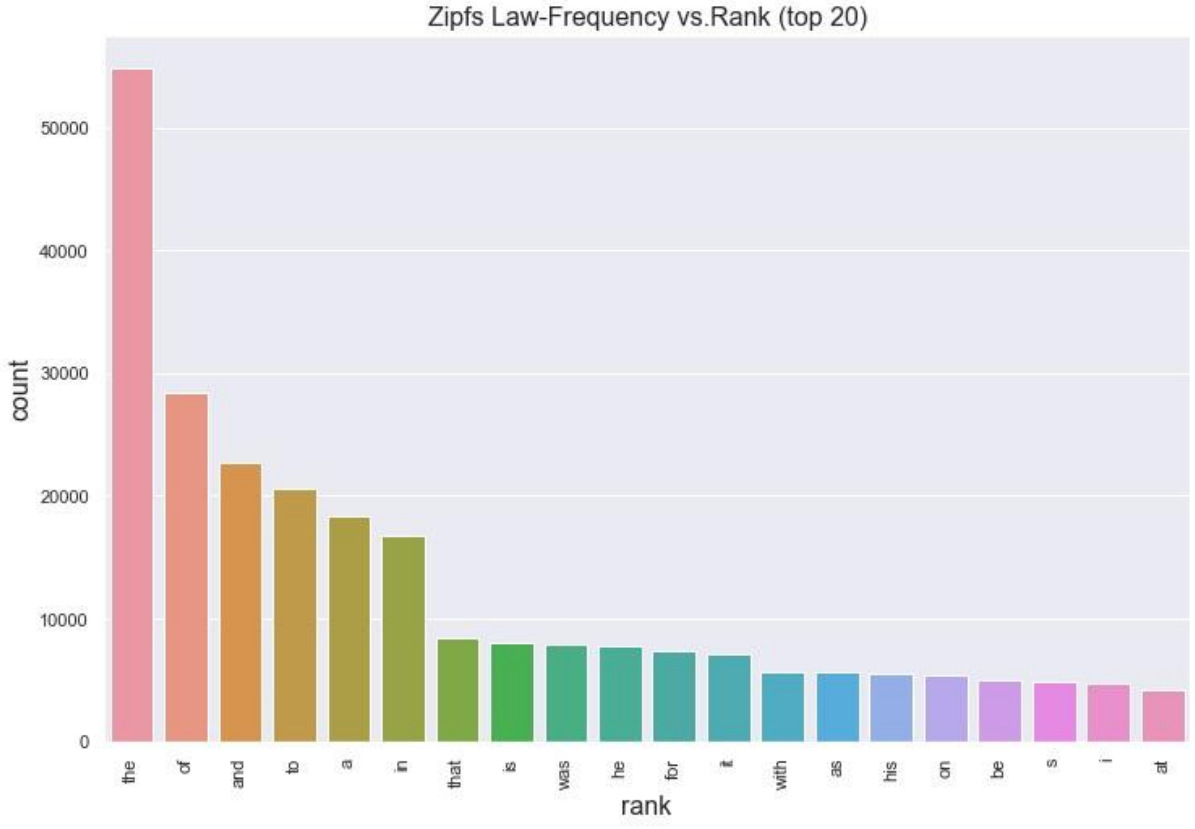
The corpus contains 99,007 lines (not sentences, but different lines), which after filtering out non alpha numeric tokens and blank lines sentences reduces to 92,574 lines.

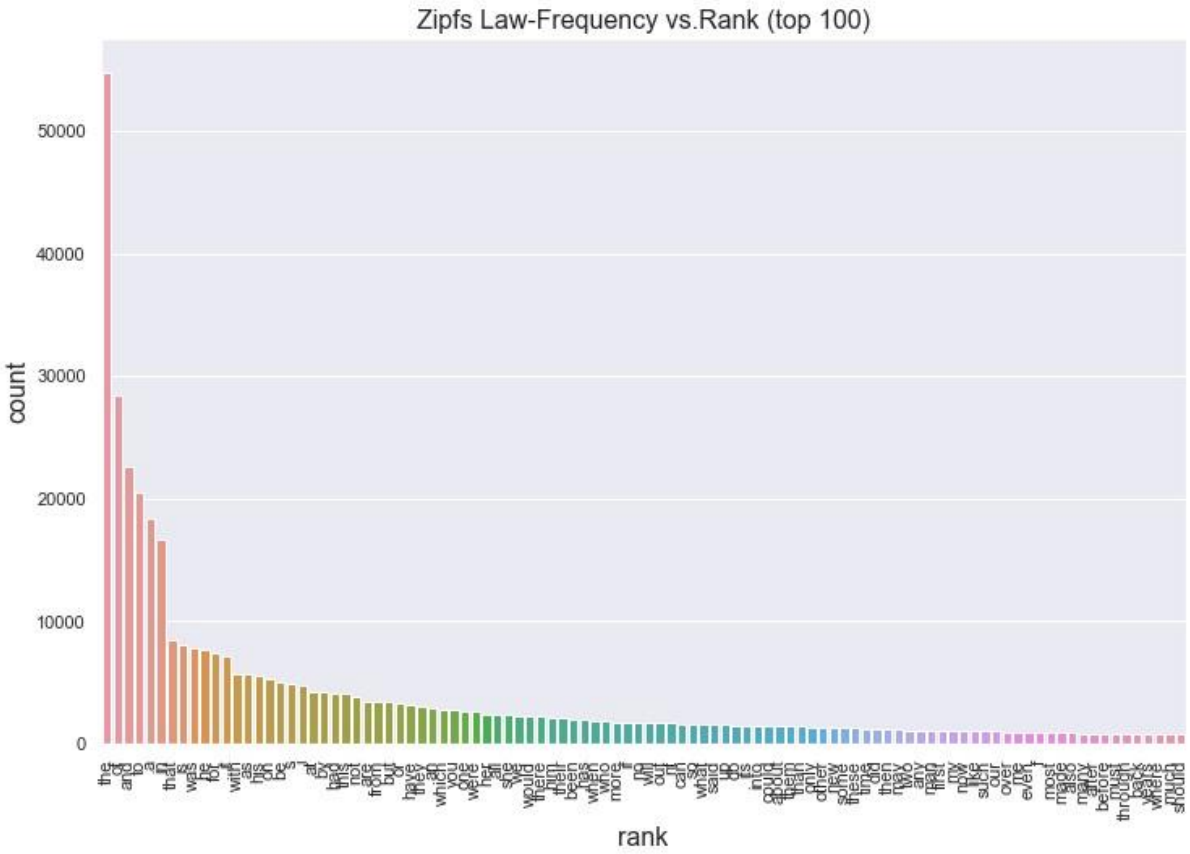
The total number of unique tokens is 42,190, and total number of tokens is 10,22,452.

A bar graph plot of the word vs its frequency of occurrence was generated for verifying if the corpus follows the Zipf's law.

Let’s see the most frequent words in order of top 10,20 and 100.







The above frequency distribution curve is in accordance with the zipf's law which suggests that the frequency of any word is inversely proportional to its rank in the frequency table.

The perplexity values shown in this report use the formula involving entropy and discrete probability distributions which is the case with our n-grams,

$$PP(p) := 2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

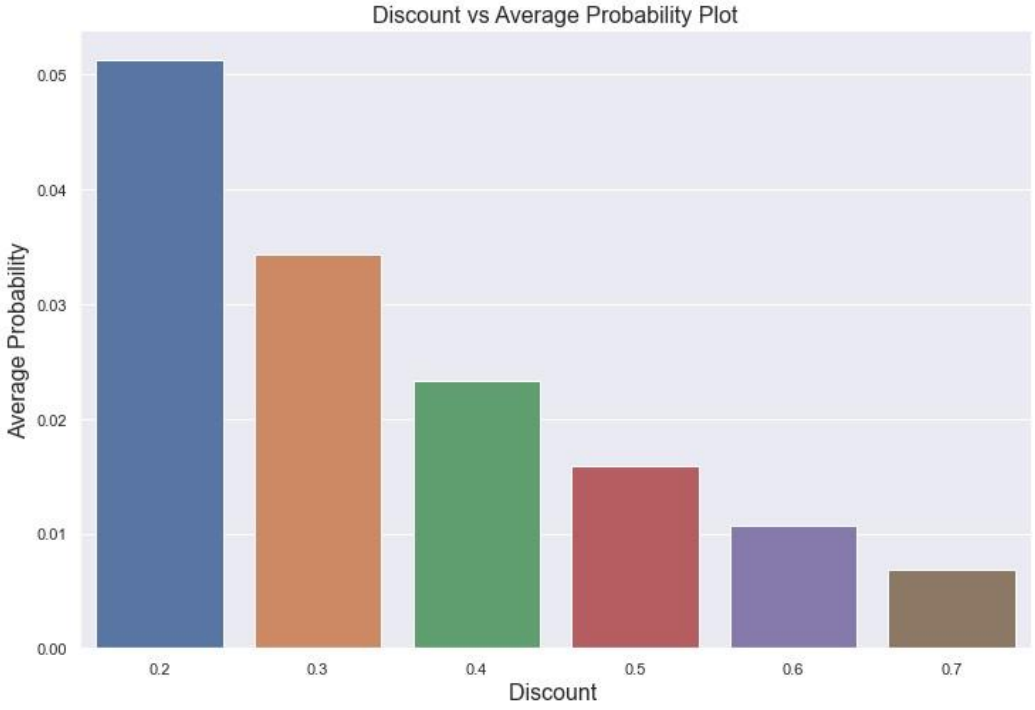
Where each P(x) is the probability of x^{th} n-gram of the sentence found by the smoothing method, H(p) is the entropy of the sentence and PP(p) is the total perplexity of that sentence.

Analysis using Kneser Ney Smoothing:

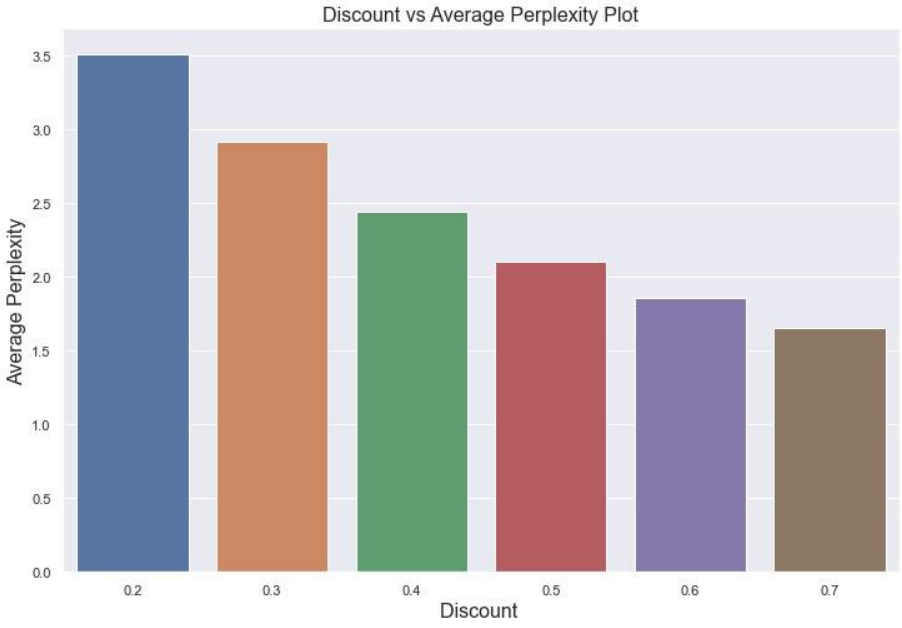
For various values of discount [0.2,0.3,0.4,0.5,0.6,0.7], the variation of average perplexity and average probability is shown below.

The results shown are for random 50 lines from the corpus.

Average Probabilities with different discounts:

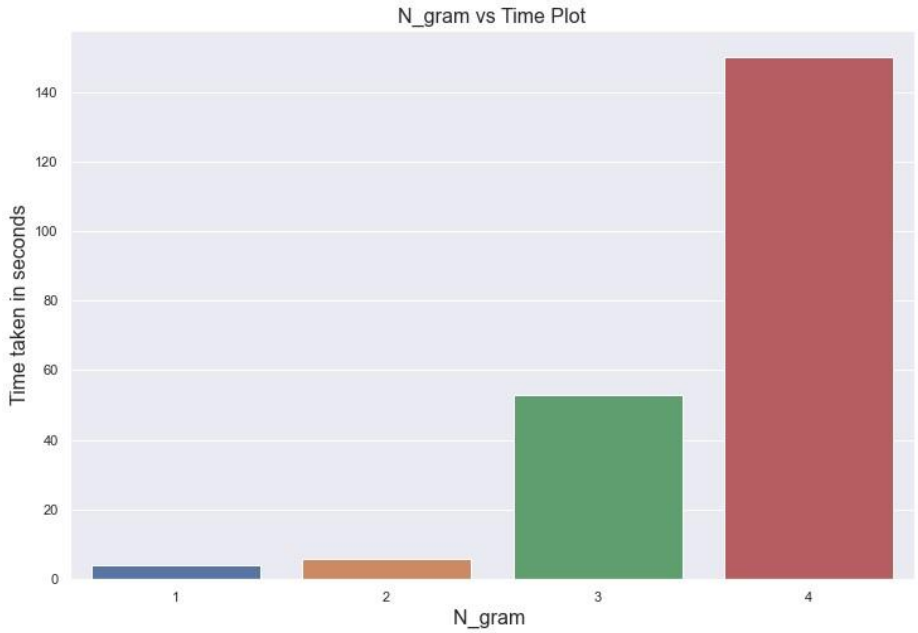


Average Perplexities with different discounts:



From the above distributions, we can see that as the discount factor in Kneser Ney increases both the probability and perplexity decreases.

For a fixed discount of 0.65, we can see the time taken for different N-Grams.



The figure shown is for randomly selected 1000 sentences from the corpus, and the N-grams range from 1 to 4.

Analysis using Witten Bell Smoothing:

For reference and comparison among statistical smoothing method for language models, Witten Bell was also compared against the Neural and Kneser Ney language models.

For Witten Bell, the average probability for 50 random sentences from the corpus was 0.028 and average perplexity was found to be 4.02.

Analysis using Neural Language model:

Now, for the same Brown corpus, a LSTM Neural Language Model was also trained.

It consisted of the following layers.

```
NeuralLM(
  (embedding): Embedding(43293, 50)
  (dropout): Dropout(p=0.5, inplace=False)
  (fully_connected): Linear(in_features=256, out_features=256, bias=True)
  (output_layer): Linear(in_features=256, out_features=43293, bias=True)
  (relu): ReLU()
  (tanh): Tanh()
  (softmax): Softmax(dim=2)
  (RNN): LSTM(50, 256, num_layers=4, dropout=0.5)
)
```

Note that this is not the actual architecture, but the names and dimensionalities of different layers.

The implementation was done in PyTorch.

i.) First the embedding is constructed for each word, the dimensionality of the embedding was set to 50 after some cross validation, but setting too large dimensionality resulted in memory error, so it was feasible to keep it at 50 and achieve good results.

ii.) The previous LSTM state and embedding outputs are then passed to LSTM units, there are 50 stacked LSTM units, each of which have a hidden size of 256.

iii.) After LSTM cells, the outputs are passed to a fully connected layer to perform better learning, both with and without ReLU activation was tested, without any activation showed better results.

iv.) After the fully connected layer, it is passed to the final output layer which has the dimensionality equal to the vocabulary size.

v.) The next sequence of words is then predicted using SoftMax function on the output probabilities.

An input batch goes through the above steps in forward propagation.

Below is the implementation of the *forward* function used for training the NLM.

```
def forward(self, inp,previous_state):
    embedding_op = self.embedding(inp)

    if self.dropout_rate is not None:
        embeddding_op = self.dropout(embedding_op)

    lstm_outputs,(h_n,c_n) = self.RNN(embedding_op,previous_state)

    fc_output = self.fully_connected(lstm_outputs)

    if self.dropout_rate is not None:
        fc_output = self.dropout(fc_output)


    predicted_word = self.output_layer(lstm_outputs)

    predicted_word = self.softmax(predicted_word)

    return predicted_words,(h_n,c_n)
```

- The optimizer used was AdamW, where the weight decay is performed only after controlling the parameter-wise step size. The weight decay or regularization term does not end up in the moving averages and is thus only proportional to the weight itself, and hence works better than Adam for generalizing models and is better suitable for NLP, gets the best of both worlds (SGD 's generalizability and Adam's speed).
- The loss function used was Cross-Entropy loss.
- Learning rate scheduler was also used to decrease the learning rate after no improvement is seen over consecutive epochs or if the model begins to overfit.

```
learning_rate = 0.01

optimizer = optim.AdamW(
    neuralLM.parameters(),
    lr = learning_rate,
)

scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=2,
                                                 threshold=1e-5, threshold_mode='rel',
                                                 cooldown=0, min_lr=0, eps=1e-08, verbose=False)

criterion = nn.CrossEntropyLoss()
```

The input to the Neural Language Model was a sequence of N-grams.

Example:

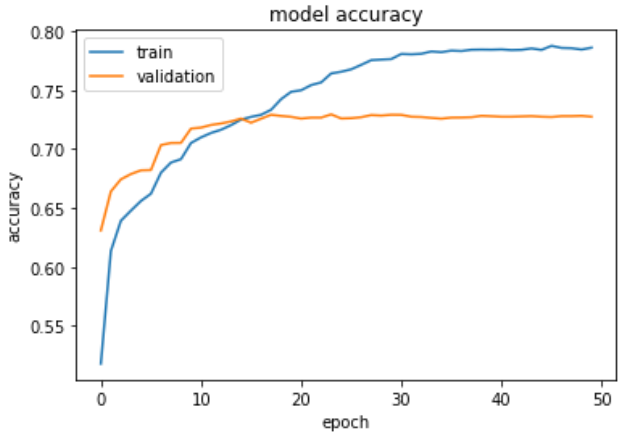
If a sentence consists of words [w1, w2, w3, w4, w5, w6, w7].

The 4-gram input and output sequences look as follows.

Input Sequence	Expected Output Sequence
1. w1, w2, w3 w4	w2, w3, w4, w5
2. w2, w3, w4, w5	w3, w4, w5, w6
3. w3, w4, w5, w6	w4, w5, w6, w7

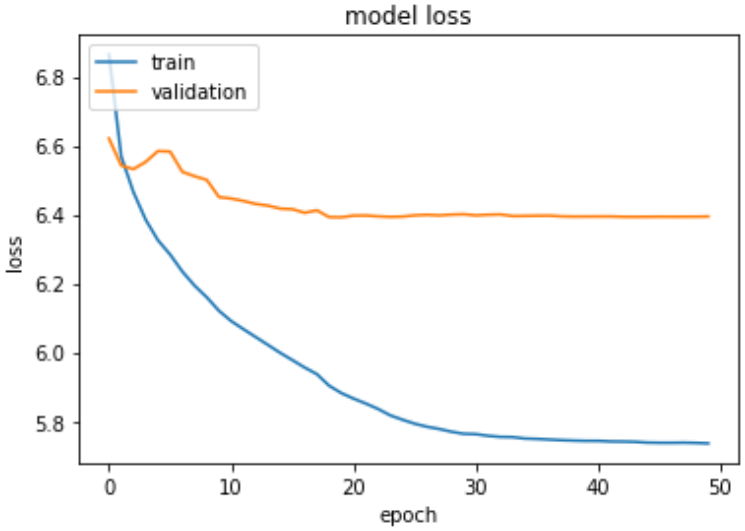
For each of the sequences the SoftMax layer returns a probability distribution, and we find out perplexity using the same probabilities for each of the n-grams of the sentence.

After training the model for 50 epochs, with a batch size of 128 and input and output sequences of 5 grams and initial learning rate 0.01, the following train vs validation accuracy and loss curve was obtained.



We see that as the training accuracy continues to increase, the validation accuracy plateaus at around 23 epochs, but since learning rate scheduler kept changing the learning rates by a factor of 0.5, model did not overfit to an extent that it would affect its generalization capability. Here the accuracy is the accuracy against predicted and actual sequence of words of 5-grams.

We see that the training loss continues to decrease while similar to accuracy the validation loss plateaus at around 23 epochs. So, it was best to stop the training at 50 epochs.



Final train and validation accuracy and loss after 50 epochs.

```
train_loss: 5.7358298729543815 train_acc : 0.7859428510273972
Validation : val_loss 6.394351330209286 val_acc : 0.7273936170212766
```

Results:

Comparison of different language models on both train and test set.

The test set consisted of 10000 sentences for each Language Model.

Metric used was perplexity, which uses the formula mentioned earlier. In case of Statistical Language models, training set is larger because no validation was done, and the models perform better if there is more training data to them.

In Neural Language model, 77,574 sentences were used for training and 5000 sentences were kept for validation.

All three models were tested on an unseen test set, of size 10,000 which was common to all.

A random state of 42 was common to all the cases for splitting into test set.

i.) Results and Comparison of performance on train set

LM Name	**# Sentences**	**Corpus type**	**Avg. Perplexity**
Witten Bell	82574	Brown	4.57
Kneser Ney(d=0.65)	82574	Brown	2.1
Neural Language Model(with ReLU activations)	77574	Brown	1.84
Neural Language Model (**without** ReLU activations)	77574	Brown	1.45

We see that the Neural Language Model without ReLU activation outperforms all the other LMs by a good factor, it shows 44% improvement over Kneser Ney, and 215% improvement over Witten Bell and 26.9% improvement over NLM with no ReLU activations.

This significant improvement can attributed to the learning capabilities of Neural Networks which can learn non-linear patterns hidden in the data. **ii.) Results and Comparison of performance on test set**

LM Name	**# Sentences**	**Corpus type**	**Avg. Perplexity**
Witten Bell	10000	Brown	1.15
Kneser Ney(d=0.65)	10000	Brown	1.1
Neural Language Model(with ReLU activations)	10000	Brown	1.852
Neural Language Model (**without** ReLU activations)	10000	Brown	1.453

We see that NLM performs a little worse than the statistical LM's, the reason might be because it was trained on a very small corpus,

Neural Networks require huge data to generalize for unseen data, this corpus only had 42000 unique words and only 1.02M tokens, moreover, even that portion went into train, validation and test splits, so according to me if the dataset had been larger, the results on unseen test data would've been significantly better.

The testing time for Neural Language model was significantly less than Kneser Ney and Witten Bell.

Conclusion:

In this report we discussed the performance of various language models and how size of dataset affects the performance of Neural Language Models.