Language Modeling on Penn Treebank Dataset

Pranjali Basmatkar
University of California, Santa Cruz
Codalab Username : Pranjali
pbasmatk@ucsc.edu

1 Introduction

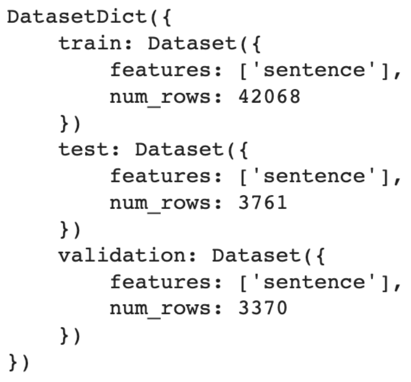
Language modeling (LM) is the problem statement of using various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. The task predicts the probability of the next word occurring. Here, given a sequence of words x(1), x(2), . . . x(t), we compute the probability distribution of the next word x(t+1).

Traditionally statistical language models have been used such as n-gram models. Here, we work on developing a neural language model (Bengio et al., 2000). These language models are based on neural networks and are considered as an advanced approach to execute NLP tasks. These models overcome the shortcomings of statistical models such as handling rare and unique words.

Language models are considered as the Super tasks which can be further used to enhance the performance of various downstream tasks (Bengio, 2008). They have been popularly used in problem statements like machine translations, sentiment analysis, speech recognition, text suggestion and parsing tools.

1.1 Understanding the data

The penn treebank dataset is available through the datasets library from huggingface. A sample snippet of the dataset structure can be seen in Figure 1.



```
DatasetDict({
    train: Dataset({
        features: ['sentence'],
        num_rows: 42068
    })
    test: Dataset({
        features: ['sentence'],
        num_rows: 3761
    })
    validation: Dataset({
        features: ['sentence'],
        num_rows: 3370
    })
})
```

Figure 1: Dataset Structure

The data comes with the following pre-processing:

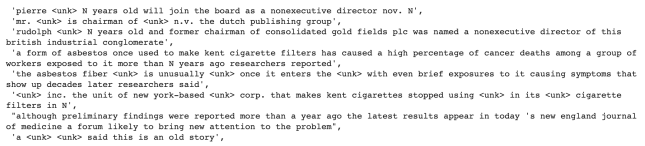
- Unknown rare words are replaced with the 'unk' token in the data

Dataset	Data points
Train	42068
Validation	3370
Test	3761

Table 1: Data Split.

- Numerical data has been replaced with 'N' token

The dataset has been divided into three sets, training, validation and test. The exact count of the datasets can be seen in Table 1. A sample snippet of the data can be seen in Figure 2.



```
 'pierre <unk> N years old will join the board as a nonexecutive director nov. N',
 'mr. <unk> is chairman of <unk> n.v. the dutch publishing group',
 'rudolph <unk> N years old and former chairman of consolidated gold fields plc was named a nonexecutive director of this
british industrial conglomerate',
 'a form of asbestos once used to make kent cigarette filters has caused a high percentage of cancer deaths among a group of
workers exposed to it more than N years ago researchers reported',
 'the asbestos fiber <unk> is unusually <unk> once it enters the <unk> with even brief exposures to it causing symptoms that
show up decades later researchers said',
 '<unk> inc. the unit of new york-based <unk> corp. that makes kent cigarettes stopped using <unk> in its <unk> cigarette
filters in N',
 "although preliminary findings were reported more than a year ago the latest results appear in today 's new england journal
of medicine a forum likely to bring new attention to the problem",
 'a <unk> <unk> said this is an old story',
```

Figure 2: Sample Data

1.2 Input-Output data creation

The Penn Treebank dataset is a raw text containing a series of sentences. To convert this data into input and output for the model to learn and validate its learning, certain steps were taken to extract the input and target sentences.

- The entire text was tokenized using space
- A sequence length was assumed and word sequences of that sequence length were extracted out of the text. eg. for sequence length 50,

  x0 : token 0 to token 50

  y0 : token 1 to token 51

  x1 : token 50 to token 100

  y1 : token 51 to token 101 and so on.

- This process of input and output data creation resulted in 7308 training samples and the entire set of input-output samples for the train, validation and test set can be found in Table 2.

Dataset	Data points
Training samples	7308
Validation samples	580
Test samples	647

Table 2: Data Samples.

1.3 Data Encoding techniques

Data encoding refers to the process of transforming textual data into a numerical form. This can be achieved using various encoding techniques. For the sake of experimentation, the entire training raw text was used to create a dictionary. This dictionary was later used to assign unique numerical identifiers to each of the tokens present in the dataset. Additionally, ”unk” (unknown tokens) and ”sos” (start of the sentence) tokens were added to the dictionary. A total of 9719 unique tokens were extracted from the dataset, and this will be further used as the vocabulary for training.

2 Model

2.1 Pre-trained embeddings

Textual data represents meaningful information based on their context, semantic relations with other words and meaning. Words in different contexts represent different information. Pre-trained embeddings are used to add context for the tokens to the model. For training the language model, the FastText pre-trained embeddings (Mikolov et al., 2018) were used.

wiki-news-300d-1M - The wiki news pre-trained embeddings are 1 Million word vectors trained on the Wikipedia 2017 corpus and statmt.org news dataset(16B tokens) having embeddings dimension of 300. The initial vocabulary of the training data was used to initialize the pre-trained embeddings. This resulted in embeddings of dimensions 9719 X 300, where 9719 is the vocab size while 300 is the dimension of the pre-trained embeddings.

2.2 Recurrent Neural Networks

Recurrent neural networks are a type of artificial neural network that was created in the 1980s (Jordan,) that is generally used for temporal problem statements such as machine translation, language modeling etc. It has cyclic connections between nodes allowing output from a node to affect subsequent input to the same nodes. For language modeling, the recurrent models use ’memory’ to utilize prior inputs to influence the current input and output. This memory allows the neural network to remember dependencies longer than the regular feed-forward networks. Despite its advantages in handling sequential data, RNNs face the issue of vanishing gradient where the value of the gradient becomes too small, and the model stops learning. This issue is addressed by LSTM (Long short-term memory).

2.3 Gated recurrent network

Gated recurrent units (Chung et al., 2014)are a type of RNN that were developed to improve the performance of the vanilla RNN. GRUs uses two gates, the update gate and the reset gate. These gates determine what information is allowed through to the output and can be trained to retain information from farther back. This allows it to make better predictions. GRUs are also known to utilize less memory and are also faster as compared to other recurrent neural networks.

2.4 Long Short term memory

Long Short-Term Memory networks (LSTM) (Hochreiter and Schmidhuber, 1997) are a type of recurrent neural network that is capable of handling sequential data while also saving long-range dependencies in its memory. LSTM have three gates as compared to the two utilized by GRUs. It has an input, output and forget gate that allow retention of long range dependencies. This functionality allows the network to gather more context and also solve the challenge of vanishing gradients faced by the RNNs. LSTMs are designed to avoid the long-term dependency problem (Chung et al., 2014). LSTMs have feedback connections, i.e., it is capable of processing the entire sequence of data.

2.5 Training set up and hyper-parameters

Once the data preparation was completed, the various hyper-parameters were identified which can be used for model tuning. Some of them are stated below.

- Sequence length: Refers to the number of tokens to be included in each sequence.
- Batch size: Number of training samples to be used in each batch
- Learning rate: The learning rate to be specified to the optimizer
- Number of Epochs: Number of epochs to train the neural network on.
- Number of layers: Number of layers in the recurrent neural network.

3 Experiments

3.1 Data splitting techniques

For the purpose of the experimentation, the Penn treebank dataset is divided into three sets. Train, validation and test. The training and validation sets were used during the training phase, and the test set was used to measure the performance of the model on unknown data. The data split provided in the dataset can be seen in Table 2.

3.2 RNN

For developing the RNN model, the layer layers were implemented

- Embeddings layer: This layer is supported by the FastText pre-trained embeddings that were previously initialized for the vocabulary of the training data. It takes the pre-trained initialized embeddings and the input text as its inputs.

Sr. No.	Model	Architecture	Seq_Len	N layers	Batch Size	LR	Epoch	Train perp	Valid perp	Test perp
1	RNN	Vocab - 512 - vocab	32	1	32	0.01	4	8574	4311	4329
2	GRU	Vocab - 512 - vocab	64	1	64	0.01	7	1585	1782	1799
3	LSTM	Vocab - 512 - vocab	64	1	64	0.01	4	66.5	117.9	115.7

Table 3: Baseline creation

Sr. No.	Model	Architecture	Seq_Len	N layers	BatchSize	LR	Epoch	Train perp	Valid perp	Test perp
1	LSTM	Vocab - 512 - vocab	64	1	64	0.01	4	66.5	117.9	115.7
2	LSTM	Vocab - 512 - vocab	128	1	64	0.01	7	62.07	113.9	105.8
3	LSTM	Vocab - 512 - vocab	256	1	64	0.01	8	65.7	115.1	118.1
4	LSTM	Vocab - 512 - vocab	512	1	64	0.01	12	71.06	123.8	124.8
5	LSTM	Vocab - 1024 - vocab	128	1	64	0.01	8	60.8	108.6	118.2
6	LSTM	Vocab - 1024 - vocab	128	2	64	0.001	21	91.3	144.2	136.7
7	LSTM	Vocab - 512 - vocab	128	2	64	0.001	16	139.7	149.3	162.8

Table 4: Tuning results

- RNN layer: This is the main RNN layer with tunable parameters like the number of layers, bi-directional approach etc. For the purpose of baselining, the number of layers was set to 1 and bidirectionality was turned off. This layer takes two inputs. Initially, a hidden state will be seeded as a matrix of zeros, along with the first input to the RNN cell.

- Linear layer: This is the output layer that receives the output of the RNN and uses the output dimensions (vocab size in this case) to predict the next token for the given sequence.

3.3 GRU

In addition to the embeddings layer and the output layer described in the RNN section, this neural network uses an GRU layer.

- GRU layer: This is the layer that the GRU functionality in the neural network. This layer takes two inputs. The hidden state seeded with a matrix of zeros along with the first input to the cell.

3.4 LSTM

In addition to the embeddings layer and the output layer described in the RNN section, this neural network uses an LSTM layer.

- LSTM layer: This is the main layer that incorporates the LSTM functionality in the neural network. For the baselining, the number of layers was set to 1 and bi-directionality was turned off. This layer takes two inputs. The hidden state seeded with a matrix of zeros has two parts, h and c which are passed as a tuple, along with the first input to the LSTM cell.

3.5 Performance Metrics

To analyse the model's performance, two performance metrics were explored. The loss and perplexity.

Perplexity: Perplexity is a popular evaluation metric for a language model. It measures how well the proposed probability model predicts a sample. In PyTorch, perplexity is calculated by exponentiation of the entropy (the loss).

3.6 Baseline selection and model tuning

Upon experimentation with the three sequential models (as seen in Table 3.), it was seen that the LSTM provided the best results. Further to improve the performance of the LSTM (see base architecture in Figure 3.) the following hyper-parameters were tuned.

- Number of layers: The number of layers 1 and 2 were experimented with. It was observed that the training epochs required by the 2-layer LSTM were considerably higher than that of 1 layer LSTM. It was also noticed that the model performed better with just one layer.

- Learning rate: Learning rates such as 0.01, 0.001. 0.005 were experimented with, and it was seen that with a learning rate of 0.01, the model was able to converge much faster and achieve a good performance.

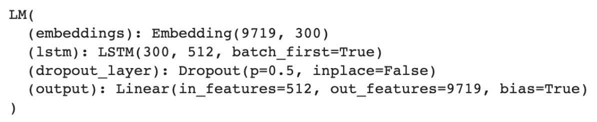
- Sequence length: Various sequence lengths were experimented with such as 64, 128, 256 and 512. It was seen that the model performed well with sequence lengths 64 and 128.

- Batch Size: Batch sizes such as 32, 64 and 128 were attempted. It was noticed that a smaller batch size resulted in a long training time and a larger batch size, like 128 and above resulted in a large memory consumption. Batch size 64 is considered optimal.

The entire set of tuning results can be seen in Table 4.

3.7 Model Tuning

Upon tuning the model using the given hyperparameters, it was seen that the LSTM model with 1 layer, 128 sequence length, 64 batch size, a learning rate of 0.01 and trained for 4 epochs resulted in the best validation score. The proposed



```
LM(
  (embeddings): Embedding(9719, 300)
  (lstm): LSTM(300, 512, batch_first=True)
  (dropout_layer): Dropout(p=0.5, inplace=False)
  (output): Linear(in_features=512, out_features=9719, bias=True)
)
```

Figure 3: Model architecture

model has a training perplexity of 62.07, and a validation perplexity of 103.9. This model was further tested on the test set and was able to achieve a perplexity of 107.7. The model tuning results can be seen in Table 4.

4 Results

Based on the analysis done during the tuning phase, the LSTM model is recommended for language modeling using sequential data. The proposed model received a training perplexity of 62.07, a validation perplexity of 113.9 and a test perplexity of 105.8 as seen in Figure 4 and 5. To further improve the performance of the model, techniques such as the attention mechanism can be experimented with.

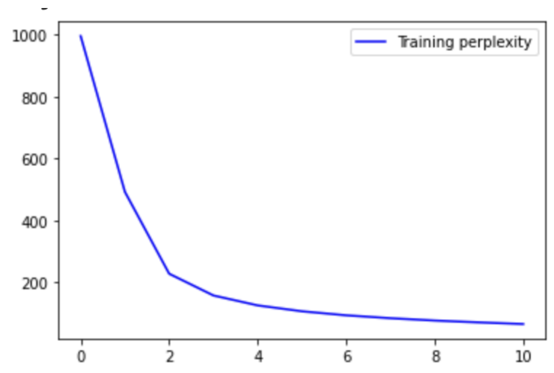


Figure 4: Training Perplexity

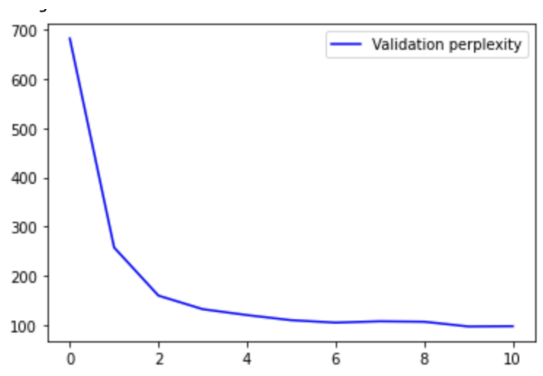


Figure 5: Validation Perplexity

References

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Y. Bengio. 2008. Neural net language models. *Scholarpedia*, 3(1):3881. revision #140963.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

M I Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.