COSC 420 ASSIGNMENT 2 – ID 7562257

TASK 1: Bengio et al – A Neural Probabilistic Language Model

This classic paper describes a neural network applied to the field of statistical language modelling. It introduces an issue present in traditional methods ("the curse of dimensionality"), explains the distributed-representation-of-words approach taken to fight this issue, and relates previous works in the field. It then describes how the neural network was applied in creating this distributed representation and elaborates on implementation details for the required performance. Finally, it reports experimental results against traditional methods and speculates on future extensions to the model.

N-grams and the curse of dimensionality

The curse of dimensionality is particularly relevant to the sentence structure of statistical language modelling, but also affects any joint distribution model of discrete random variables. In this field, words are the discrete variables, with a large but finite number of values in any language's vocabulary. The probability of any word occurring in a sentence is dependent on the surrounding words, so a joint distribution that gives the conditional probability of a word given its predecessors is modelled.

While words at the opposite end of sentences can correlate in syntactic structures (for example, the plural form of "The *dog*, though quiet, *does bite*" and "The *dogs*, though quiet, *do bite"*), the curse of dimensionality comes when traditional methods are limited in their scope to model these correlations. This prediction of long syntactic structure needs to occur in new sentences, which are not observed in the input set of sentences known as the corpus.

Part of the reason these limitations occur in sentence modelling is due to the sheer size of combinations where all words in a vocabulary and all sentence lengths are concerned. Generalizations can be more easily obtained in other large modelling tasks with continuous variables (such as decimal numbers) because the function to be learned can be expected to have local smoothness over continuous variable values. Conversely, discrete variable functions are difficult to generalize – most observed words or characters in a large corpus will have a near-maximal hamming distance from each other regardless of how they were encoded, and will have a drastic impact in the estimated function value if changed. This means that modelling across whole sentences is too great a task for many traditional methods, and generalizations cannot be made easily to reduce the scale of the task.

One such traditional approach to statistical language modelling is N-grams, where word combinations of a context length (usually n = 3) in the corpus are counted, and those counts used to generate conditional probabilities. They rely on the increased statistical dependence of nearby words to model language efficiently using this limited context window, however as stated above this limited window of dependence does not always apply. Thus, the issue taken with N-grams in this paper is their susceptibility to the curse of dimensionality, from a limited context window and a reluctance to generalize over discrete word values.

Proposing a Neural Network

A Multi-Layer Perceptron is proposed to overcome the curse of dimensionality. This neural network learns simultaneously a vector representation of the vocabulary, and the parameters for the joint probability function.

$$f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$$

Figure 1 - function to be learned by the network, decomposed into probability function g and word vector mapping C

The learned vocabulary distribution is in the form of *word feature vectors*, which represent words as points in a vector space and encode semantic and syntactic similarities. “Running” and “walking”, for example, would be close together in the vector space. Probability learned from one’s frequent appearance in the corpus would be shared to the other in the trained model, for example, causing them to both appear frequently in suitable locations in the predicted text. This generalization across previously discrete variables can occur in the proposed model due to a continuous distance in vector space representing word similarity in the learned word vector mapping .

Unlike an N-gram, the number of free parameters only scales linearly with the vocabulary size and the context window size. The context window, and therefore the number of inputs to the network, can be increased with effective results to overcome the curse of dimensionality without excessive run-time (especially with the parallel implementation proposed). In the results of the paper, it is commented that increasing the context window from 2 to 4 words improved the neural network, but not the n-gram. This is due to “each training sentence inform[ing] the model about a combinatorial number of other sentences”.

Evaluation

The learned model was evaluated against a number of N-gram models on the metric of test perplexity (normalized probability of the whole corpus). Low perplexity indicates high probability for the corpus, a marker of an effective estimation of the joint probability function. The best N-gram model had 24% greater perplexity than the best neural network.

A Modern Comparison

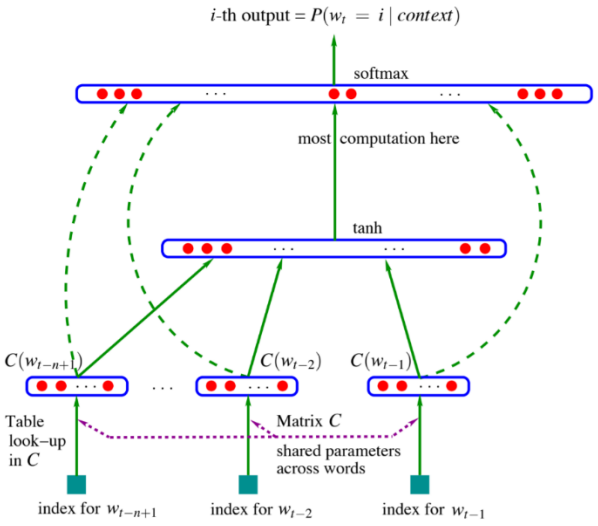

Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Figure 2 - network architecture from Bengio et al, page 6 of pdf

The Multi-Layer-Perceptron used in this paper differs in structure to more recent Long Short-Term-Memory networks used in this report. A MLP, as observed in the graph, has a definite direction of calculation from the context window inputs to the conditional probability output. Unlike an N-gram, the number of free parameters only scales linearly with the vocab size and the context window size. Bengio et al suggest a recurrent neural network could be used to make this scaling sub-linear.

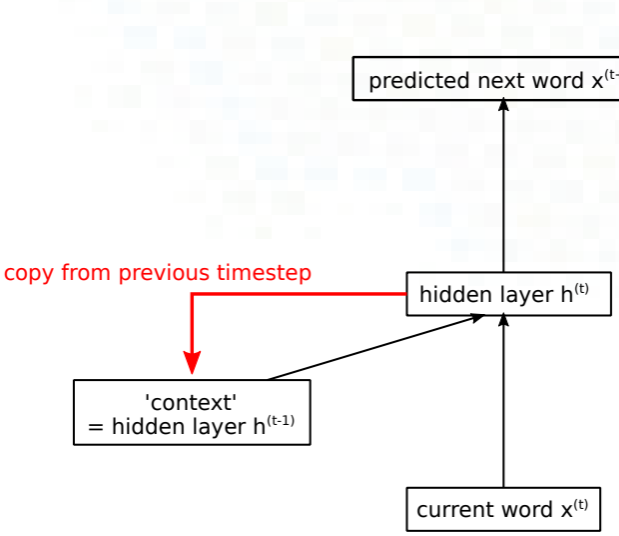

Figure 3 - Recurrent Neural Network diagram, COSC 420 Lecture 9

The defining feature of RNNs is their use of a copy of the network at the previous timestep, which retains a "memory" of previous words/characters that decays exponentially. LSTM networks, like those implemented in task 2, go one step further in preserving past word context by separating long-term memory of past learning into a memory cell, while short-term memory of the current context and text prediction are done by the hidden layer(s).

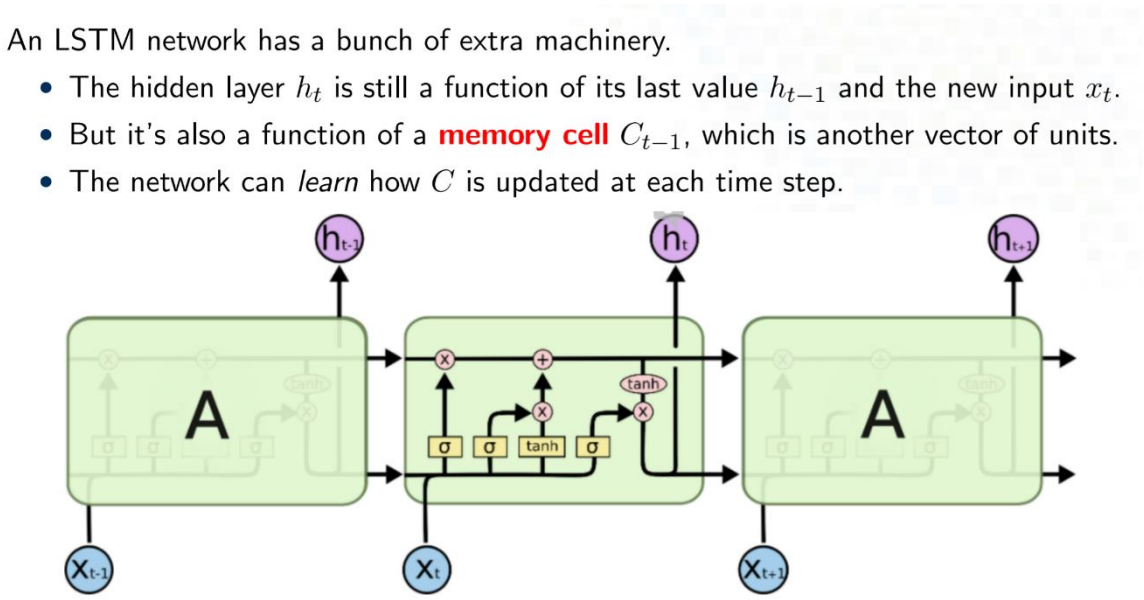

Figure 4 - Long Short-Term Memory network diagram, COSC 420 Lecture 10

TASK 2 – Recurrent Neural Networks

The word and character-based models implemented were trained on the provided corpus of Shapespeare's works and based off Jason Brownlee's examples:
https://machinelearningmastery.com/how-to-develop-a-word-level-neural-language-model-in-keras/
https://machinelearningmastery.com/develop-character-based-neural-language-model-keras/

Due to memory issues with the whole corpus being loaded into memory at once, approximately the first quarter was used for both character and word models. 1,135,884 char sequences and 207,959 word tokens were found in this corpus, with a vocabulary of 12,851 unique word tokens and 65 chars.

```
raw_text = doc_methods.load_doc("shakespeare_input.txt")
# CUT FILE TO 1/4 SIZE
raw_text = raw_text[:int(len(raw_text) / 4)]
```

Placing the whole document into a tf.Dataset object and the fit_generator method were also explored to input the complete Shakespeare works, but working implementations could not be found.

Using a random selection of the most likely characters returned from model.predict(), rather than the singular prediction of model.predict_classes(), was explored to combat word repetition loops in the character model, but could not be successfully implemented.

In general, few of the suggested changes were made to the Brownlee code to better generate Shakespeare-esque data, due to difficulties in finding documentation. The Keras documentation has significant gaps (such results from their internal search engine not showing on the page and depreciated but workable functions such as fit_generator being completely absent). The Keras examples found elsewhere were difficult to draw syntax and usage conclusions from due to poor quality of explanations or difference in subject matter, and the environmental setup did not allow me to find method argument data from my IDE.

Text Generated

Character-level model on *input chars*:
Romeo: I would they shall be the state and the service to
. Consider to the such a son of the sun shall be the state a
, but that the state and the service to the such a son of th
Hello world to the such a son of the sun shall be the state

Word-level model in *input words:*
romeo excused edward is the lark and wild and look down
consider now the frozen malkin disorderd spring ingrateful lends the narrow
but that thy pent fully killed his riches upon the crown and
hello world leased oer meat hew on the drunkard swayed it from

Word output on a 50 word seed text, with posthumous formatting:
[…] heart int and now farewell till half an hour hence.
Ferdinand: a thousand thousand.
Prospero: so glad of this as they i cannot be who are surprised withal but my rejoicing at nothing can

be more ill to my book for yet ere suppertime must i perform much business appertaining.
stephano: come come on me and northumberlands eternal mercury hath met

Perplexity

Perplexity was calculated as the exponential of the cross-entropy loss of the network: this was observed from the final training printout.

Character level at 50 epochs, 10 + 1 context window, batch size: 32
cross-entropy loss: 1.3945, accuracy: 0.5787, perplexity: 4.03296

(note that a larger context window of 20 + 1 characters was tested, but lead to no significant decrease in perplexity or increase in casually observed accuracy of generation for the increased training time)

Word level at 50 epochs, 50 + 1 context window, batch size: 64
cross-entropy loss: 3.3924 - accuracy: 0.2767, perplexity: 29.7372

TASK 3: Word and Character-level Models

Both RNN models take files of overlapping context windows as input, with the character-level model taking 10+1 chars and the word-based model taking 50+1 word tokens.

The word-level model puts these 50 (punctuation-free) inputs sequences through an embedding layer to define the representation of words, two LSTM layers with 100 cells each, a fully connected layer interpreting the features extracted, and a fully connected softmax layer to output the probability distribution.

The character model puts these inputs with 10 time steps through a LSTM layer with 75 cells and a fully connected softmax layer to output a probability distribution for the vocabulary.

The key differences between these models, aside from the inputs, is the addition of an embedding and hidden layer either side of the deeper Long Short-Term Memory layers. This is because the word-level model, like the Bengio et al model, learns a word embedding alongside the probability distribution of that embedding.

Both networks have significantly lower perplexities than the Bengio et al MLPs (with perplexities across train and test sitting in the hundreds). While conclusions cannot be directly drawn from the lack of experimental design, the advancement of the field since 2003 informs us the RNN and LSTM structure is mainly responsible for this improvement.

Perplexity is significantly lower for the character-level model, suggesting it has better joint probability function produced than the word-level model. I believe the word-level model, however, is a better model of human language.

The character-level model, particularly on longer generated sequences, is biased towards repeating frequent words in Shakespeare's works and their close relatives ("to the such a son of the sun shall be the state" with such, son and sun for example). While this gives an insight into Shakespeare vocabulary, it does not give much meaningful syntactic structure to the generated lines. Although the character-level model does take punctuation into account, the text generated rarely includes it, so offers no advantage in comprehension over the word-level mode. The character-level model

could be of more use when extended beyond the Brownlee tutorial, but in its current form shows little accuracy beyond the small (e.g. “and the service to”) structures repeated in its text.

The word-level model shows more evidence of syntactic structures in its generated text, such as nouns being connected to verbs and adjectives in a logical (if nonsensical in meaning) way. The phrase “*but that* thy pent fully killed his riches upon the crown, and…”, with punctuation added, seems to be a complete chunk of Shakespeare-esque dialogue.

Note that he choice of corpus is significant when comparing models. Shakespeare’s works are not in written sentence structure, unlike most of the Brown corpus or news archives used in Bengio et al. His lines run on and repeat words for emphasis in a spoken word structure, and rely heavily on commas for meaning. While Bengio et al include punctuation in their word-level model, the recurrent word model lacks it, and the character-level model possibly has too few occurrences in the corpus causing its absence in the results.

Future improvements on the two RNNs would include input formatting and context window refining to further suit the corpus text, implementing whole-corpus input and more parameter testing if working on faster hardware, implementation of prediction sampling to overcome the repetition issue in character-level models, and possibly experimenting with other corpuses to model more punctuation or compare more fairly with the Bengio et al paper.