BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

VISUALIZING NEURAL NETWORK USED AS A LANGUAGE MODEL

VIZUALIZACE NEURONOVÉ SÍTĚ POUŽITÉ JAKO JAZYKOVÝ MODEL

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JAKUB RYŠÁNEK**
AUTOR PRÁCE

SUPERVISOR **Ing. KAREL BENEŠ,**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

148495

Institut:	Department of Computer Graphics and Multimedia (UPGM)
Student:	**Ryšánek Jakub**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**Visualizing Neural Network Used as a Language Model**
Category:	Artificial Intelligence
Academic year:	2022/23

Assignment:

1. Get acquainted with language modeling and neural networks
2. Get acquainted with methods for visualization of neural network behaviour
3. Design and implement methods for inspecting processes of interest in a neural language model
4. Demonstrate them on a suitable language model and text data

Literature:

- Ian Goodfellow and Yoshua Bengio and Aaron Courville: Deep Learning

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Beneš Karel, Ing.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	6.5.2023

Abstract

Long short-term memory (LSTM) network is a type of neural network designed to analyze sequence data. The advantage of LSTM over the simple recurrent neural network is the ability to store long-term dependencies, which allows them to reach higher accuracy when performing tasks such as speech recognition or language modeling. However, due to their complexity, the internal processes that lead to these results are still not fully understood. To explore their inner workings, I created three visualization methods. These methods focus on the pattern of the behavior of the single unit present in the model or the behavior of the whole model when processing words with similar syntactic or semantic meanings.

Abstrakt

LSTM sít je typ neuronové sítě, která je určena na analýzu sekvenčních dat. Výhodou LSTM oproti jednoduché rekurentní neuronové síti je schopnost ukládat dlouhodobé závislosti, což umožňuje dosahovat vyšší úspěšnosti při provádění úloh jako je rozpoznávání řeči nebo jazykové modelování. Avšak vzhledem z jejich komplexitě není zcela jasné jak přesně fungují. Abych prozkoumal jejich vnitřní chování tak jsem vytvořil tři vizualizační metody. Tyto metody se zaměřují na vzor chování jednotlivých prvků modelu nebo na chování celého modelu při zpracování slov s podobným syntaktickým nebo sémantickým významem.

Keywords

visualization, neural networks, LSTM, clustering, t-SNE

Klíčová slova

vizualizace, neuronové sítě, LSTM, shlukování, t-SNE

Reference

RYŠÁNEK, Jakub. *Visualizing Neural Network Used as a Language Model.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Karel Beneš,

Rozšířený abstrakt

Rychlý rozvoj neuronových sítí přinesl zásadní změny ve zpracování přirozeného jazyka. V poslední době se také rozšířilo povědomí o jazykovém modelování především kvůli vydání API s GPT-3 a GPT-4 modely. Pro jazykovém modelování jakožto základní úlohu při zpracování přirozeného jazyka je klíčové pochopené dlouhodobých závislostí. Tyto závislosti obsahují informace o kontextu celého text což při uchování jejich poznání vede ke kvalitnějšímu modelování jazyka. Proto byly vytvořené neuronové sítě určené zejména pro uchovávání dlouhodobých závislostí jako jsou například LSTM sítě. Tyto sítě obsahují buňky, které se zaměřují na uchovávání informací o předešlých stavech sítě. Kombinací těchto informací a aktuálního vstupu je LSTM schopno generovat přirozeně znějící texty. I když je LSTM schopno generovat přesvědčivé texty, tak stále nejsou známy jeho vnitřní procesy, které k těmto výsledkům vedly.

Pro přiblížení těchto procesů jsem vytvořil vizualizační metody, které popisují chování jednotlivých částí LSTM neuronové sítě. Jedna vizualizace je založená na aktivacích neuronů při analýze textu. Objevil jsem, že i když většina neuronů nemá specifické vzory ve svých aktivacích a jejich pravděpodobnostní rozdělení aktivací se podobá Laplaceově distribuci, tak jsou v modelech přítomny i neurony, jejichž aktivace je unikátní pro určitou vlastnost právě analyzované části textu. Tyto vlastnosti byly nejčastěji spojené s tím, že aktuálně analyzovaný text se nacházel mezi uvozovkami či závorkami nebo tehdy když model předpovídal, že následující token bude číslo. Pomocí odlišení od pravděpodobnostní distribuce průměrného neuronu jsem byl schopen vyfiltrovat tyto neobvyklé neurony. Pro snadnější pochopení vzoru aktivací tohoto neuronu jsem vytvořil vizualizaci, která převádí hodnotu aktivace na slově, které se aktuálně nachází na vstupu neuronové sítě na modrou a červenou barvu. Tato vizualizace může být použita ať už pro demonstraci specifické aktivace u vyfiltrovaných neuronů, tak i pro demonstraci jaká byla modelem očekávaná pravděpodobnost vyskytujícího se slova.

Další vizualizace se soustředila na reprezentaci slova ve vysoko dimenzionálním prostoru, tím že je definované aktivacemi neuronů v jednotlivých vrstvách a stavech. Cílem bylo demonstrovat, jak jednotlivé části modelu reprezentují slova, která mají společnou vlastnost. Tato vlastnost může představovat například podobný sémantický či syntaktický význam nebo jejich pozice v textu. Pomocí této metody jsem demonstroval chování charakteristické pro dílčí části modelu se dvěma skrytými vrstvami. V první vrstvě jsem demonstroval, že shluky slov jsou tvořeny na základě sémantického nebo syntaktického významu což platilo, jak pro skrytý stav, tak pro stav buněk (cell state). Ve druhé vrstvě v cell state jsem objevil, že slova jsou shlukována, jako po sobě jdoucí sekvence slov představující společný kontext.

Výstupem práce jsou tyto pozorování a programy pomocí, kterých je možno vizualizovat LSTM sítě.

Visualizing Neural Network Used as a Language Model

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Karel Beneš. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jakub Ryšánek
May 10, 2023

Acknowledgements

I would like to thank my supervisor, Ing. Karel Beneš for guidance and providing valuable feedback.

Contents

Chapter 1

Introduction

The rapid development of neural networks has revolutionized natural language processing (NLP). Recently, there has also been a widespread awareness of language modeling as a fundamental task in processing, primarily due to the release of APIs with GPT-3 and GPT-4 models. For high-quality language modeling, it is crucial to understand long-term dependencies. These dependencies contain information about the context of the entire text. Remembering these dependencies leads to higher-quality language modeling. This is why neural networks, such as LSTMs, have been designed to store long-term dependencies. These networks contain a cell state that focuses on storing information about the previous states of the network. By combining this information and the current input, the LSTM is able to generate natural-sounding text. Although LSTMs have the potential to generate high-quality text, the internal processes that lead to these results are still unknown.

In this thesis, I have developed visualization methods to describe the behavior of the different parts of the LSTM neural network. The first visualization is based on the activations of the neurons during text analysis. This method aims to filter neurons whose activations have a specific pattern which should dependent on the currently analyzed text. Next, I am able to visualize this behavior by converting the magnitude of the activations into a color. This method can be used to demonstrate a specific activation of the filtered neurons as well as to demonstrate what was the expectancy of the following words by the whole model.

Another visualization is based on the representation of the word in a high-dimensional space defined by the activations of neurons in each layer and state. The goal was to demonstrate how the different parts of the model are able to cluster words that have a common property. This property may represent similar semantic and syntactic meaning or their occurrence in the text.

Chapter 2

Language modeling and neural networks

Language modeling uses mathematical techniques to determine the probability of a given sequence of elements where the elements usually represent words, letters, or parts of words (subwords). A dataset used for training and evaluating a language model is divided into three parts – train set, validation set, and train set. A test set is used for measuring the quality of a model. Measure to evaluate the quality of a model is called perplexity, and the formula for calculating perplexity is given by

$$PP(S_N) = P(S_N)^{-\frac{1}{N}} = \sqrt[N]{\prod_{k=1}^{N} \frac{1}{P(s_k|s_0 s_1 \dots s_{k-1})}}$$

where S_N is a sequence of values with the length N. There may occur a situation where the test data contains an element that is not in the training data. This word is then called unknown or out of vocabulary (OOV). The percentage of OOV words is called the OOV rate, and these words are labeled as pseudo-words called <UNK>. There are two ways of handling <UNK>. The first one is to have a fixed vocabulary, and every word that is not contained is set to <UNK>. The second possibility is to create a vocabulary implicitly and replace words with <UNK> based on their frequency. One method is to replace every word that occurs less than x times by <UNK> or choose a fixed length of vocabulary N, and only N most frequent words are not replaced by <UNK>.

2.1 n-gram language model

The simplest type of language model is the n-gram. The n-grams are analyzing a sequence of elements of length n. According to the size of n, the n-grams are divided into subtypes such as a unigram where $n = 1$. The words have no relationship among themselves. Each word has a specific value which is determined by its frequency in the training data. For a bigram where $n = 2$, each pair of words has its value. These tuples and their corresponding values are stored in a dictionary. The sum of all frequencies must equal to 1. The probability of occurrence of the examined part of the text of length m is then calculated as

$$P(w_1, \dots, w_m) = \prod_{i=2}^{m} P(w_i|w_{i-(n-1)}, \dots, w_{i-1})$$

For bigram, it would mean that the probability of the sentence

$$S = [\text{“The”, “weather”, “is”, “nice”}]$$

would be calculated as

$$P_s = P(\text{“The”}|\text{<start>}) \cdot P(\text{“weather”}|\text{“The”}) \cdot P(\text{“is”}|\text{“weather”}) \cdot P(\text{“nice”}|\text{“is”})$$

where the <start> token represents the start of a sequence. The weakness of this language model is that a single element is associated with a fixed-length sequence of elements, which does not always capture the context of the text. The larger the n and the larger the training data, the higher the accuracy and the lower the perplexity. However, with this comes problems with dictionary size. The length of a dictionary depends on the n in n-gram and on the size of the vocabulary and is calculated as $D = V^n$ where D represents dictionary and V represents vocabulary. For this reason, the biggest n-gram that is used often is trigram. The next problem occurs when a sequence of words is not contained in the training set but takes place in the test set. Then the probability of this tuple and the probability of the whole sentence would be zero. The bigger the n, the more frequently this situation happens.

Smoothing

It is important not to assign zero to a context that was seen in the test set but was not seen in the train set because then the probability of the analyzed text would be zero. To ensure that there is a modification called smoothing [2]. This modification is implemented in several ways. The most basic one of them is Laplace smoothing. This algorithm adds one to all the n-gram counts before normalizing them into probabilities. In a unigram the probability of words is defined as

$$P_L(w_i) = \frac{c_i + 1}{N + V}$$

where c_i is the original number of occurrences in the text, N is the number of words in the text and V is the size of the vocabulary, which is added because V number of words was increased. Laplace smoothing does not perform well in n-gram models, but other smoothing algorithms use similar concepts. For instance, an algorithm called add-k smoothing is on the same basis only instead of adding one adds k. Another method for smoothing is called backoff and interpolation. The idea behind backoff is that if a probability of $P(w_N|w_{N-1}w_{N-2})$ for trigram is computed and the probability of this event is zero, then the probability is estimated by using the bigram probability. The same process is applied for the bigram representation by unigram if, in bigram, the probability of a sequence is also zero. So the backoff only uses lower n-gram if there is zero probability for a higher n-gram. Whereas the interpolation method always combines probability from the original n-gram and the lower n-grams. By using the interpolation method the trigram $P(w_N|w_{N-1}w_{N-2})$ probability is calculated as

$$\hat{P}(w_N|w_{N-1}w_{N-2}) = \lambda_1 P(w_N) + \lambda_2(w_N|w_{N-1}) + \lambda_3 P(w_N|w_{N-1}w_{N-2})$$

where $\lambda_1 + \lambda_2 + \lambda_3$ is equal to one. Every λ can be chosen or can be computed by conditioning on the context.

2.2 Neural network language model

Even though n-grams are simple and it is possible to easily train them on very large datasets, the performance is limited by its simplicity. There is no connection between the words over the longer sequence because and therefore, the context of the text is lost. Because of that, a different approach to the word representation was presented in [1]. The idea is to represent a word by a vector which is called word embedding. This language model is harder to train than n-gram model but, on the other hand, provides better results. This approach of a word represented by a vector is often used in deep learning because it is suitable for the neural network, which takes a vector of the embedding size of the word as input. The thought behind word embedding is that words with similar meanings should be close to each other in the vector space. With this dependency, when a sentence "The cat is walking in the bedroom" appears in the train set, it should be possible to generate a sentence "A dog was running in a room" with similar probability. That is because the words in the first sentence have similar semantics as the words in the second sentence. This feature is also not present in the n-gram model. Then it was shown that it is possible to

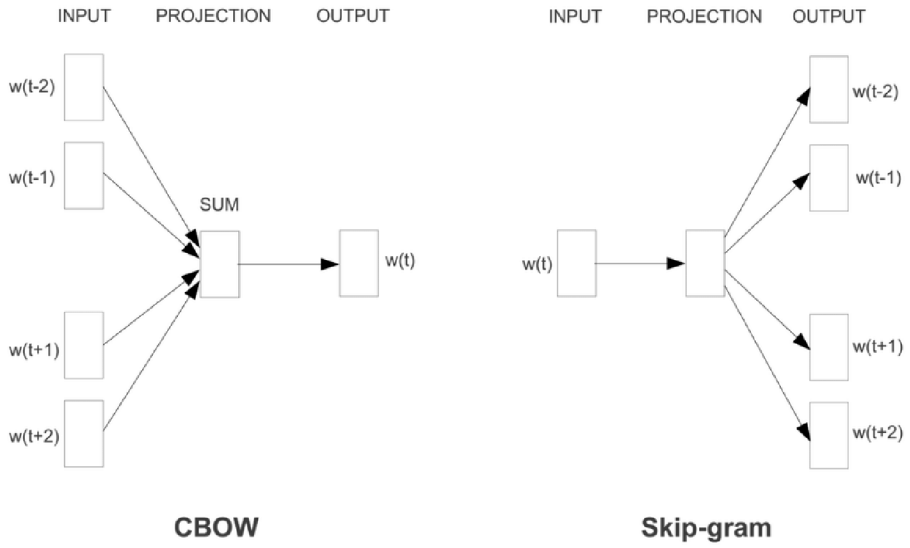


Figure 2.1: Bag-of-words (CBOW) model predicting current word from the context and Skip-gram model predicting context from the current word [7].

perform simple algebraic operations on the word embeddings. For instance, the operation $vector(\text{“}London\text{''}) - vector(\text{“}England\text{''}) + vector(\text{“}France\text{''})$ should result in a vector that is closest to the vector representation of the word Paris. Two architectures for learning distributed representations of words were proposed in [7]. The first architecture is a bag-of-words model, which predicts the current word is based on the previous words in the following words. The best performance was seen with four words in the history and in the future. The second architecture is a Skip-gram model. In contrast to the bag-of-words model, it predicts previous words and following words based on the current word as shown in Figure 2.1. By combining the Bag-of-words model and Skip-gram model, they created a new implementation of word embedding called word2vec.

2.3 Artificial neural network

An artificial neural network is a computational model used in machine learning. It is based on nodes called neurons and on the connections between them. The neurons form layers, which are divided into hidden layers, an input layer and an output layer. Connections between neurons are not within the layer, so the neural network is often visualized only by layers. The input is converted to the output through the hidden layers where all calculations are performed.

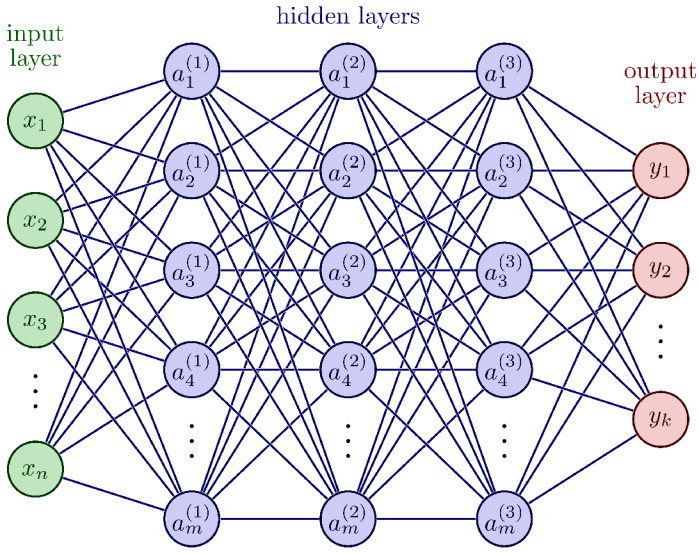


Figure 2.2: Structure of a neural network[1].

All input data have to be converted into a vector so that they can be processed by a neural network. In Figure 2.2 x_i represent input neurons, $a_i^{(k)}$ are neurons in hidden layer and y_i are output neurons. The value of a neuron in the hidden layer or output layer $a_j^{(L)}$ is calculated as

$$z_j^{(L)} = (\sum_{i=1}^{n} w_i^{(L)} a_i^{(L-1)}) + b^{(L)} \qquad a_j^{(L)} = f(z_j^{(L)}) \tag{2.1}$$

where L is the layer in which the neuron is, $a_i^{(L-1)}$ stands for values of neurons in the previous layer, $w_i^{(L-1)}$ are weights on connections and n for a number of neurons in the previous layer. b is bias and is used for regulating neuron activation. Function f is an activation function whose purpose is to add non-linearity so the output is not linearly dependent on the input. For the lower computational cost, the values of neurons in the next layer are computed by matrix multiplication:

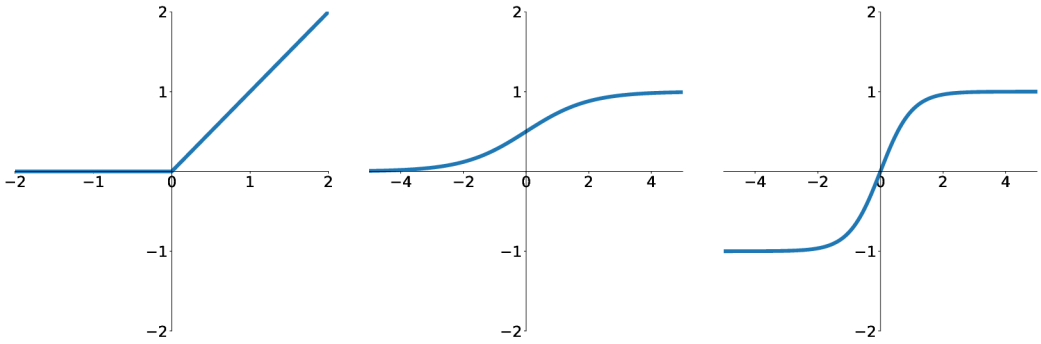
$$\vec{a}^{(L-1)} \cdot \vec{w}^{(L)} + b^L = \vec{z}^{(L)}$$

Numbers in the superscript represent the layer, $\vec{w}^{(L)}$ and $\vec{a}^{(L-1)}$ are vectors of weights and neuron activations from the previous layer. Vector $\vec{z}^{(L)}$ represents the values of neurons before applying the activation function.

[1]This picture was taken from https://tikz.net/neural_networks/

2.3.1 Activation function

An activation function is a fundamental element of neural networks. It is a nonlinear function that is applied to the value of a neuron. If the function had not been used, then the output would be linearly dependent on the input, which would make solving advanced tasks ineffective. The non-linearity allows neural networks to learn to solve advanced tasks. A common characteristic of activation functions is that they should not extensively increase the computational complexity and should retain the distribution of data.



(a) ReLU (b) Sigmoid (c) Hyperbolic Tangent

Figure 2.3: Comparison of activation functions.

The sigmoid function and hyperbolic sigmoid were used in the early days to bring the non-linearity to the model. The sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and hyperbolic sigmoid

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

bound the output in the range <0, 1> and <-1, 1> as shown in Figure 2.3 instead of all real numbers like in linear funtion. Even though this prevents the activation from blowing up, there is a vanishing gradient problem where parameters become close to zero and that leads to a minimal update in training. Another downside is the computational complexity which tries to solve an activation function called Rectified Linear Unit (ReLU). ReLU is defined as

$$\mathrm{ReLU}(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} = \max(0, x)$$

ReLU function suffers from a problem called dying ReLU which occurs when the value of a neuron is negative. Then its activations are zero, and the neuron will not contribute to the calculations. This could decrease the ability of a model to train properly. Many other activation functions with similar properties are derived from ReLU to avoid this problem.

2.3.2 Training of neural networks

Parameters in a neural network are weights and biases. These parameters are updated during training to improve the performance of the model. At the beginning of training,

the parameters of the neural network have random values based on the seed. With these random values, the output will be far from desired output. To change that, the parameters of the neural network have to be optimized based on how the generated output differs from desired output. This difference is quantified by the cost function.

Cost function

The cost function measures the error of a model over the training dataset, whereas a loss function measures the error per observation. The output of the loss function is the difference between values predicted by the neural network and correct values. There are many types of loss functions, one of them being the Cross-entropy loss function. To explain cross-entropy loss it is convenient to explain entropy and cross-entropy first.

Entropy and cross-entropy

Assume there are n mutually exclusive events, each with a different probability p_i of happening within interval <0,1>. *Surprise* is a value quantifying how unexpected is that an individual event occurs and is calculated as

$$S = \log_2 \frac{1}{p_i} = -\log_2 p_i$$

where the base of a logarithm is not mandatory to be 2[2]. This formula implies that the less likely an event is to happen, the greater the *surprise*. The expression $-\log_2 p_i$ is also known as information and is expressed in bits. The entropy $H(p)$ of p is defined as

$$H[p] = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where $-\log_2 p_i$ is a *surprise* of the event multiplied by probability of the event. The highest entropy, therefore, highest uncertainty, for any n events is when every event has the same probability of $\frac{1}{n}$. Cross-entropy measures the difference between two probability distributions and is given by

$$H[p, q] = -\sum_{i=1}^{n} p_i \log_2 q_i$$

where p_i is true distribution and q_i is estimated distribution. The closer these distributions are to each other, the fewer bits are needed. Relation between entropy and cross-entropy is $H[p] \leq H[p, q]$

The cross-entropy loss is a widely used loss function, especially for classification problems. In the case of the cross-entropy loss function, the relation between the perplexity a the cost function is defined as

$$PP(S) = P(s_1 s_2 \dots s_N)^{-\frac{1}{N}} = 2^{H(S)}$$

where the $PP(S)$ is a perplexity and $H(S)$ is the cost function on the sequence S. The result of the cross-entropy loss function is the cross entropy of probability output by a model and reference probability and helps during training to update the parameters of the

[2]If the base of the logarithm is two, then the result is in bits, and if the base is e then the result is in nats.

neural network in the right way. To control the update of the parameters neural network uses a hyperparameter called the learning rate.

Learning rate

In a decently trained model, the parameters are already close to optimal values, and if the update is too high, the values will skip over the optimal values, which would lead to deterioration in model quality. That is why a learning rate is chosen at the beginning of the training. A lower learning rate leads to training over more epochs because changes in parameters are smaller. On the other hand, if the learning rate is too high, the final state of weights and biases is often suboptimal. Common practice is to adapt the learning rate during training which helps to achieve optimal results. An example of this approach could be that the learning rate decreases as the number of epochs[3] increases. Use of learning rate while updating weights and biases:

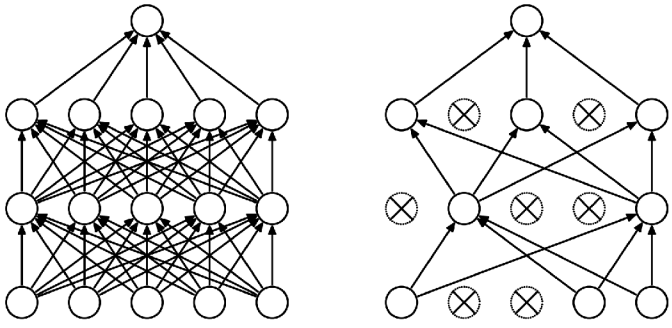
$$w_{t+1} = w_t - \epsilon \frac{\partial C}{\partial w_j^{(L)}}$$

$$b_{t+1} = b_t - \epsilon \frac{\partial C}{\partial b^{(L)}}$$

ϵ represents the learning rate, t is training time. The $\frac{\partial C}{\partial b^{(L)}}$ and $\frac{\partial C}{\partial w_j^{(L)}}$ elements are gradients, and they define how should the parameters be updated to ensure the decrease of the cost function result. In theory, this process of updating parameters would be done after every single training data is processed. However, a calculation of a gradient is very demanding for computational power. Therefore in practice, data are divided into batches and only after the whole batch is processed are parameters updated.

2.3.3 Dropout

Neural networks with a large number of neurons tend to overfit to the training data. The overfitted model performs well on the training data but poorly on unseen data. To address this problem, a hyperparameter called dropout was presented in [10]. During training, each neuron is randomly dropped, and the calculations are performed without them.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.4: Comparison of normal neural network and neural network after applying dropout [10].

[3] An epoch is done when the model iterates through all the training data.

The dropout is applied repeatedly at each training iteration, with different neurons being dropped each time. After dropout is applied, the neural network is thinned and consists of neurons that survived dropout as shown in Figure 2.4. This thinned neural network is then sampled and trained, which avoids relying on a single neuron. The dropout is turned off during evaluation or during the execution of the task so the whole neural network contributes.

2.3.4 Limitations of feedforward neural networks

Feedforward neural networks can be used for a variety of tasks where the goal is to map an input to the output. For instance, a common task for feedforward neural networks is classification, where the input is independent on the history. However, the limitations of feedforward neural networks can be seen when working with sequential data. This is caused by the fact that they treat the input and output independently, so they cannot grab the context and, therefore, correctly assess the output. Recurrent neural networks have been developed to overcome these limitations.

2.4 Recurrent neural network

Recurrent neural networks (RNN) are mainly used to process data in a sequence and are formed by multiple iterations of the simple neural network as shown in Figure 2.5. The main idea of RNN is that it can incorporate the previous state when processing current input. That is realized by sending a hidden state that carries information about context to the following iteration of RNN. The hidden state is calculated as

$$h_t = f(w_i \cdot x_t + w_h \cdot h_{t-1} + b)$$

where f is the activation function, t is the time step, x_t is the current input, w_i and w_h are weights for the input and for the hidden layer and b is bias. Since h_t is dependent on h_{t-1} then recursively, the hidden state includes traces of all previous hidden states. In each iteration, the output is calculated as

$$y_t = g(w_y \cdot h_t + b)$$

where w_y are the weights of the output layer and g is the activation function for the output layer.

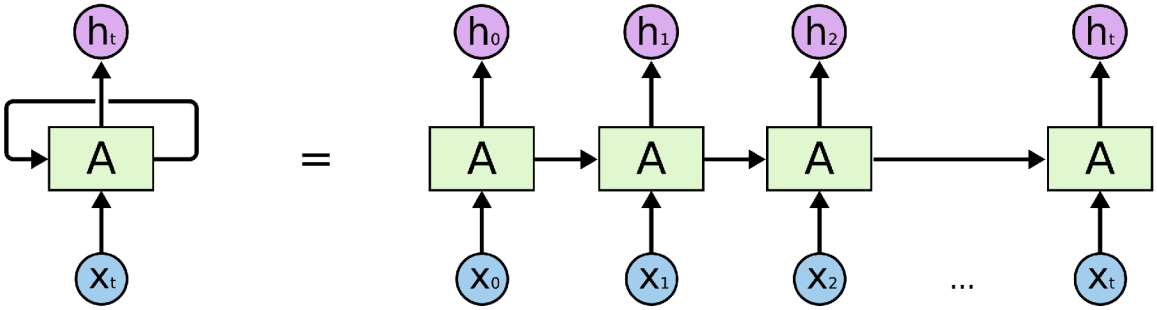


Figure 2.5: The general structure of a recurrent neural network, where h_x are outputs and x_y are inputs of the individual parts of the RNN. A are the individual iterations of the RNN that process the current input and hidden state from the previous iteration [9].

2.4.1 Problems of the RNN

Because of the way that the RNNs are structured, the gradients used to update weights and biases are propagated through a long chain of connections between the input and the output. Because of the long sequences that the recurrent neural networks processed during training, they encounter with gradient problems.

Vanishing gradient problem

During backpropagation, a gradient is calculated by multiplying a derivative of the activation function. When the derivative consists of low values, then the gradients become too small, and the update of weights and biases will be minimal, which makes the training process ineffective. This behavior is characteristic of the sigmoid activation function as shown in Figure 2.6 because the maximum value is 0.25. The simplest solution is to use an activation function that does not have a small derivative like ReLU or functions derived from ReLU described in Section 2.3.1. Another solution is using connections to earlier layers that do not go through the activations function, and thus the derivative is not applied. This approach is used in residual neural networks [3].

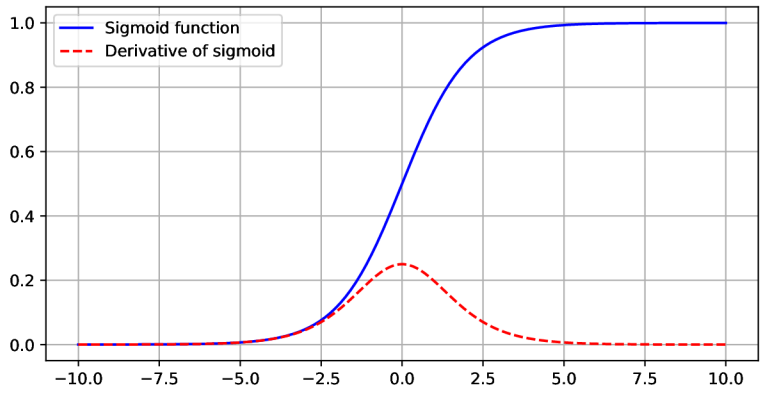


Figure 2.6: Sigmoid function with its small derivative causing vanishing gradient problem.

Exploding Gradient Problem

The exploding gradient problem is the right opposite of the vanishing gradient problem. This problem occurs when the gradient of the loss function becomes very large. Weights are repeatedly multiplied by large values, which would lead to the instability of a model. One of the solutions is gradient clipping introduced in [8]. This method is based on two approaches. Clip by value will ensure that values cannot be greater or lower than the chosen clip values. The problem here is that after clipping by value, the ratio of the values is changed and the gradient will have a different direction. The other method, called clip by norm, scales gradients between defined values. The gradient clipping by norm is defined as

$$g_{clip} = \begin{cases} g_{max}\frac{g}{||g||} & \text{for } ||g|| > g_{max} \\ g & \text{otherwise} \end{cases}$$

where g is gradient vector, $||g||$ is the norm of the gradient vector and the g_{max} is the maximum allowable norm for thw gradients. Here the ratio remained the same, but some values are too small and will not contribute to the parameter tuning.

2.5 Long short-term memory

Long short-term memory neural network (LSTM) was introduced in [5]. They were explicitly designed for remembering information in the long term and for avoiding gradient problems. Every RNN has its repeating module, and in RNN this module is very simple. However, in LSTM, this module has four layers instead of one and is shown in Figure 2.7.

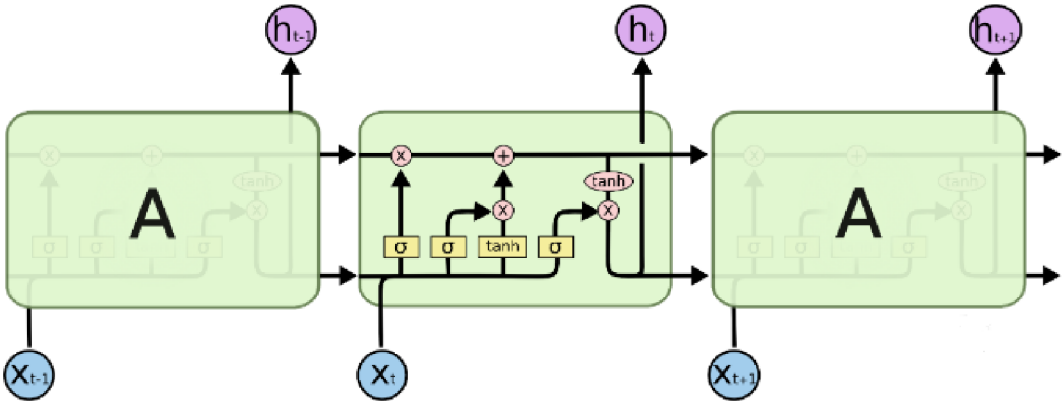


Figure 2.7: Repeating module in LSTM [9].

The main characteristic that distinguishes LSTM from the other RNNs is the cell state. Information in this part has only minor linear interactions and therefore collects information throughout the whole text. The ability to change cell state is made through structures called gates. The first gate is defined as

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where the h_{t-1} is the hidden state from the previous step, x_t is an input vector, W_f and b_t are weights and biases. This layer is called forget gate layer because it makes the decision about which information to preserve. This is caused by the multiplication of the cell state with the outputs of the sigmoid function. The next step is to store new information in the cell state which is performed by

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_C)$$

New information consists of a sigmoid layer and a tanh layer. The sigmoid layer decides which values should be updated and tanh creates a vector of new values. The new cell state is then calculated as

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t)$$

Finally, the calculation of the output is made by applying the sigmoid function to the hidden state h_{t-1} from the previous layer and current input to decide which parts are to be retained. The result is then multiplied by the hyperbolic tangent of the cell state, which carries information from the previous context:

$$h_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \cdot \tanh(\tilde{C})$$

Dropout in LSTM

That dropout used for feedforward neural networks does not work well with RNNs. The reason is that the current hidden state contains information about the previous context. By applying dropout in the hidden state, the information about the context is disrupted. This noise is then amplified by the recurrence, which hurts learning. The solution presented in [11] is to apply dropout only on non-recurrent connections as shown in Figure 2.8

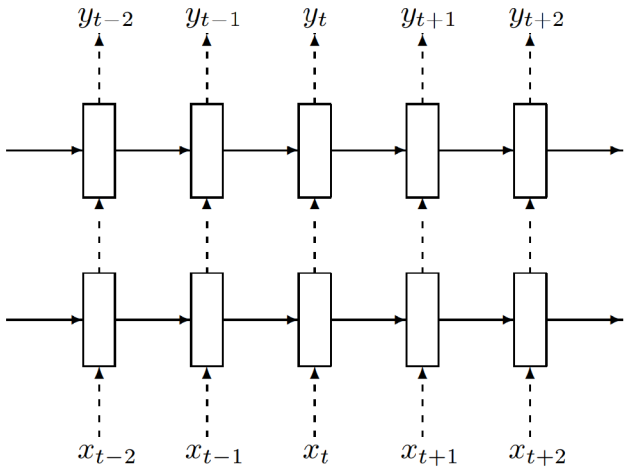


Figure 2.8: Dashed arrows indicate where the dropout is applied and the solid lines indicate where dropout is not applied [11].

Chapter 3

Visualizing high dimensional data

To visualize high-dimensional data, it is convenient to convert them into low-dimensional form (2-dimensional or 3-dimensional). The first reason is that from 2-dimensional visualization, it is possible to get more information by observing that from high-dimensional visualization. And then, it is easier to manipulate low-dimensional data by using methods such as clustering.

3.1 Clustering

Clustering is used often used in data analysis to group unlabeled data with similar properties into clusters that are easier to work with. Data can be clustered under various conditions. Therefore there is no general clustering algorithm, and a selection of a clustering method is influenced by the desired condition. One of them is the hierarchical clustering algorithm.

Agglomerative hierarchical clustering

Agglomerative clustering is a type of hierarchical clustering. Algorithm has a bottom-up strategy which means that it starts by considering every object as a cluster. At each step, two clusters that are closest to each other are combined into one. This procedure is repeated until there is the desired amount of clusters. A measuring method and linkage function are needed to recognize which two clusters are the closest. The classical measuring methods for distance measures are Euclidean and Manhattan distances:

$$d_{eu}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad d_{ma}(x,y) = \sum_{i=1}^{n}|(x_i - y_i)| \tag{3.1}$$

The linkage function takes information about the distances between clusters and connects them based on the linkage criterion. The most common linkages are single, complete, average, centroid and ward.

- single link – uses the distance between the closest elements in clusters
- complete link – uses the distance between the farthest elements in clusters
- average link – uses an average of all pairwise distances in clusters
- centroids – uses the distance between means of elements in clusters

- ward – pretends to merge two clusters, finds the centroid of the new cluster, calculates the sum of square distances between the centroid and elements and merges two clusters whose sum is the lowest

3.2 Reducing the dimensionality of high dimensional data

The process of reducing the dimensionality of high-dimensional data is about creating counterparts of high-dimensional data points in low dimensions. These counterparts need to preserve the dependencies among other data points as well as it was among original data points. However, this task is not always possible due to "Crowding problem" that is described later in this section.

Stochastic Neighbor Embedding

SNE algorithm was presented in [4]. At first, SNE converts the high-dimensional Euclidean distances between datapoints into conditional probabilities $p(j|i)$. This probability represents that x_j would pick x_i as a neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i. Probability $p(j|i)$ is dependent on the distance between points, where $p(j|i)$ is relatively high for points close to each other. The conditional probability $p(j|i)$ is given by

$$p(j|i) = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{i \neq k} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

where σ_i is the variance of the Gaussian that is centered on datapoint x_i and is set by hand or found by binary search, so the entropy of the distribution over neighbors is equal to $\log k$. Here k refers to the number of local neighbors or the perplexity[1] of SNE. Perplexity is chosen at the beginning and its typical values are between 5 and 50.

For the low-dimensional counterparts, y_i and y_j of the high-dimensional data points x_i and x_j , it is possible to compute a similar conditional probability $q(j|i)$. When the variance is set to $1/\sqrt{2}$ the computation of $q(j|i)$ is given by

$$q(j|i) = \frac{\exp(-||y_i - y_j||^2}{\sum_{i \neq k} \exp(-||y_i - y_k||^2}$$

Setting the variance differently only rescales the final map. The aim of the SNE is to find low-dimensional data representation that preserves the properties of high-dimensional counterparts. This is achieved by matching the distributions $p(j|i)$ and $q(j|i)$ as well as possible by minimizing the cost function. The method to measure differences between probability distributions is the Kullback-Leibler divergence.

Kullback-Leibler divergence, also known as relative entropy, quantifies the difference between chosen probability distribution from a reference probability distribution. It is is a non-symmetric metric $D(P||Q) \neq D(Q||P)$ calculated as

$$D_{KL}(p||q) = -\sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i}$$

[1]This perplexity does not refer to the perplexity used in language models and neural networks

where p_i is reference probability. The relationship of KL divergence, entropy, and cross-entropy would represent reference probability, and KL divergence would be $D_{KL}(p||q) = H(p,q) - H(p)$. The cost function for SNE is a sum of Kullback-Leibler divergences between the $p(j|i)$ and $q(j|i)$ probability distributions for each object:

$$C = \sum_i D_{KL}(P_i||Q_i) = \sum_i \sum_j p(j|i) \log \frac{p(j|i)}{q(j|i)}$$

For minimalization of the cost function is used a gradient descent method

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (y_i - y_j)(p(j|i) - q(j|i)) + p(i|j) - q(i|j))$$

which can be interpreted as a sum of forces applied on y_i. The magnitude of force applied on y_i by y_j is defined by the distance and by $(p(j|i) - q(j|i)) + p(i|j) - q(i|j))$ element.

t-Distributed Stochastic Neighbor Embedding

Even though SNE constructs good visualizations, the weak spots of SNE are that the cost function is challenging to optimize and an occurring problem called The Crowding problem. The goal of t-SNE introduced in [6] is to reduce the impact of these problems. SNE forms the basis for t-SNE. The cost function used by t-SNE is a symmetrized version of the SNE cost function with simpler gradients and Student-t distribution instead of a Gaussian to compute the similarity between two points in the low-dimensional space. The difference between these two distributions is shown in Figure 3.1

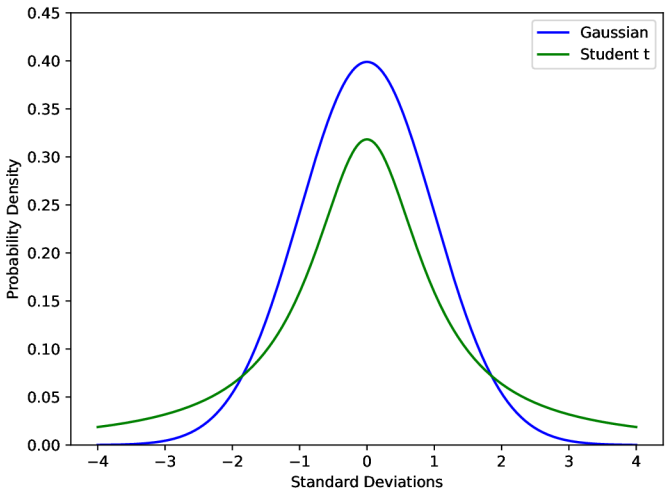


Figure 3.1: Comparison of Gaussian distribution and Student t-distribution with one degree of freedom.

The "t" in t-SNE stands for Student-t distribution, and by using a heavy-tailed[2] distribution in the low-dimensional space to oppose the crowding problem.

[2]distributions that are heavier-tailed have less significant decline as shown in 3.1

Symmetric SNE

To counter the problem with the minimizing cost function, t-SNE uses single KL divergence between joint probability P and Q instead of the sum of KL divergences s between the conditional probabilities $p(j|i)$ and $q(j|i)$.

$$C = D_{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

In symmetric SNE, the pairwise similarities in the high-dimensional space p_{ij} are defined as

$$p(j|i) = \frac{\exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{i \neq k} \exp(-||x_i - x_k||^2/2\sigma^2)}$$

The formula for p_{ij} in the symmetric SNE is almost the same as the formula for $p(i|j)$ in normal SNE. The only difference is that the variance for high-dimensional space is the same for all high-dimensional points. Therefore it has the property that $p_{ji} = p_{ij}$ and $q_{ji} = q_{ij}$. The main advantage of the symmetric SNE is the simpler form of its gradient, which is faster to compute:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (y_i - y_j)(p_{ij} - q_{ij})$$

The crowding problem

The crowding problem lies in the fact that there are cases where it is not possible in a two-dimensional map to accurately model distances between points in higher dimensions. In n-dimensional space, it is possible to have $n + 1$ mutually equidistant points. That would mean that the probability distribution over those points is uniform. There is no way to project this characteristic in two-dimensional space. This problem occurs when data points that are far apart in the high-dimensional space are mapped to the points that are very close to each other in the lower-dimensional space. In t-SNE a Gaussian distribution is used in the high-dimensional space, and in low-dimensional space is used a probability distribution with heavier tails than a Gaussian, which is Student t-distribution. This allows t-SNE to preserve the local structure of the data while spreading out the clusters to reduce the crowding problem.

Chapter 4

Implementation

For visualizing the expectancy of the word, I used PIL library to create a picture with the current word and then selected a background based on the expectancy, where red color stands for high and blue colors stands for low expectancy. Then I aligned these words based on their position in the text. The example of the output is in Figure 4.1.

Figure 4.1: Visualization of the expectancy of the next word in training data, where red color indicates a high probability and blue color indicates low probability.

This visualization is used not only for the prediction of the whole neural network but also for individual neurons and their activation.

Next is the visualization of the neuron activation through analyzed text. The experiment consists of two programs. First one outputs values of all neurons in all layers. The second program analyzes outputted values and visualizes them in the bar chart. Visualization was achieved with the numpy and matplotlib libraries. One neuron consists of n values where n is a number of words in the analyzed text. These values are divided into intervals with a length of 0,01. This way, the activations are represented as a probability distribution. I used the probability distribution of neuron activations to distinguish neurons that have characteristics of average neurons by applying Kullback-Leibler divergence.

The final visualization is performed on the activations over one word. This way, a word is represented by n values where n is the number of neurons in a single layer. To visualize the n-dimensional word in two dimensions, I used t-SNE. Then I applied agglomerative hierarchical clustering on the 2-dimensional data to group words that were close in n-dimensional space into clusters. This experiment also consists of two programs where the first one takes the activations over the words in analyzed text and saves them into

a dataframe, applies t-SNE, and uses agglomerative hierarchical clustering with single or ward linkage, and these values also save into the same dataframe. The second program filters values for possible unwanted clusters a visualizes it with the scatterplot function from the seaborn library.

Tool for cluster visualization

Then I created a tool in a python notebook for visualizing clusters from the experiment described above. The visualization is represented in Figure 4.2. The tool is programmed in jupyter notebook with ipywidgets library, and it takes values from the dataframe that is created by the program processing values with t-SNE.

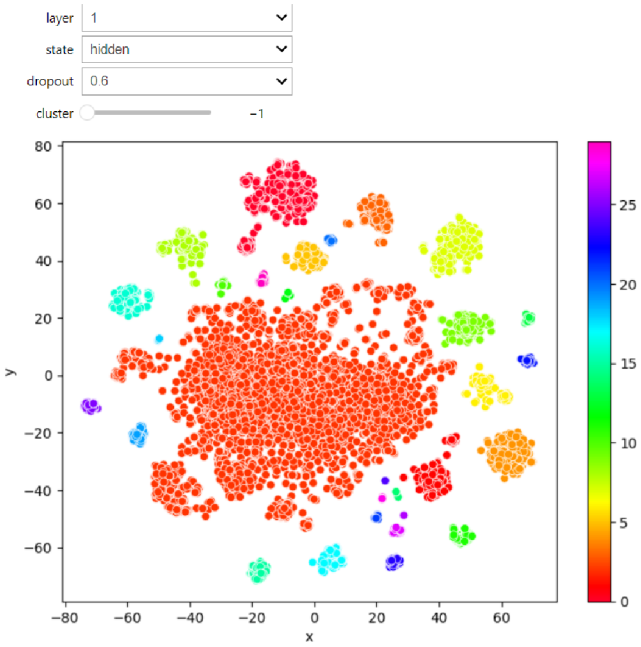


Figure 4.2: Possible state of visualization tool.

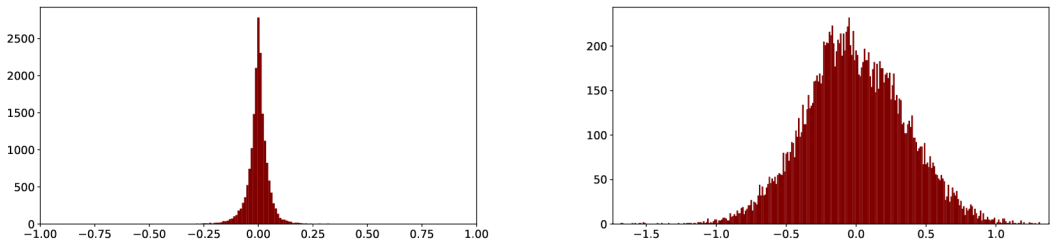
Chapter 5

Experiments

I trained 9 LSTM models[1] with the same parameters except for seed and dropout. The dropout for the network ranged from 0 to 0.8. Models have various dropouts so I could compare the results and declare what is the impact of the dropout. They have two hidden layers with a size of 2000, and the size of word embeddings is 400. Their perplexity is between 90 and 140 based on the dropout. All neural networks were trained on wikitext-2. That is a dataset with a vocabulary size of 33 thousand words.

5.1 Values of a neuron

I expected that in the trained neural network should be some neurons that have a certain characteristic in their activations. Some could be activated when the end of a sentence is coming or when the words are between brackets. So I divided the activations of neurons into the parts of size 0.01. After the activations are categorized, it is possible to think of these values as a probability distribution. In the hidden state, neurons have values between -1 and 1, and the cell state does not have fixed limits (Figure 5.1).



(a) Hidden state

(b) Cell state

Figure 5.1: Comparison of neuron activations in hidden state and cell state.

The majority of neurons do not indicate any special behavior, and the neurons in the hidden state usually have a shape of Laplace distribution with the center in 0. However, there are neurons that do not resemble the Laplace distribution. The shape of these neurons takes various forms, as shown in Figure 5.2. This neuron is from the hidden state in the

[1]model, train/valid/test data, and the training script of LSTM on Wikitext-2 were taken from https://github.com/pytorch/examples/tree/main/word_language_model

first layer and has a shape of half of a Laplace distribution in the positive values, but in the negative values, the activations are mostly close to -1.

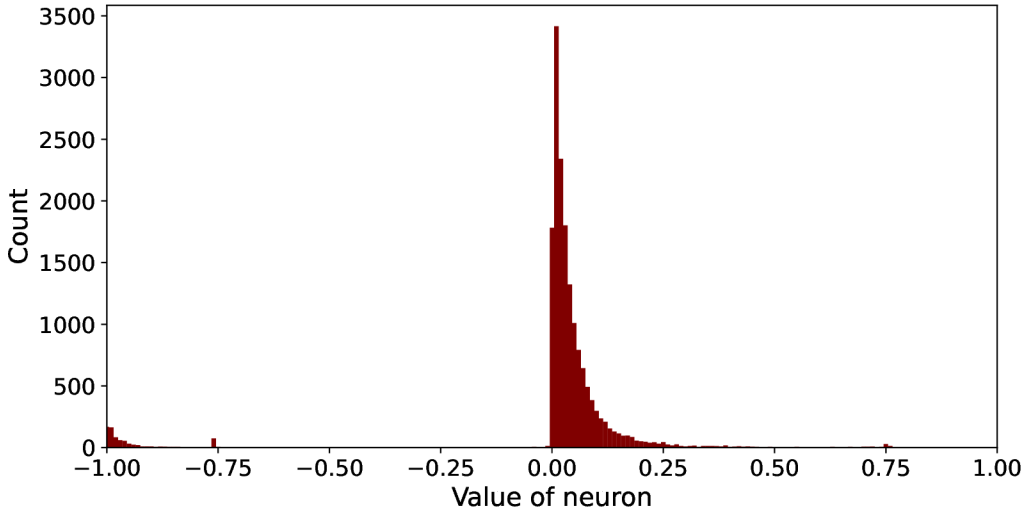


Figure 5.2: neuron with high amount of activations close to -1 value.

In the hidden state of the first, these values clearly separated from the rest, indicate a reaction on a recurring pattern in analyzed text. Specifically, the neuron in Figure 5.2 has high activations when the word on the input is between quotation marks visualized in Figure 5.3. Activation of the neurons that indicate the quotation marks have the values separated the most from the rest.

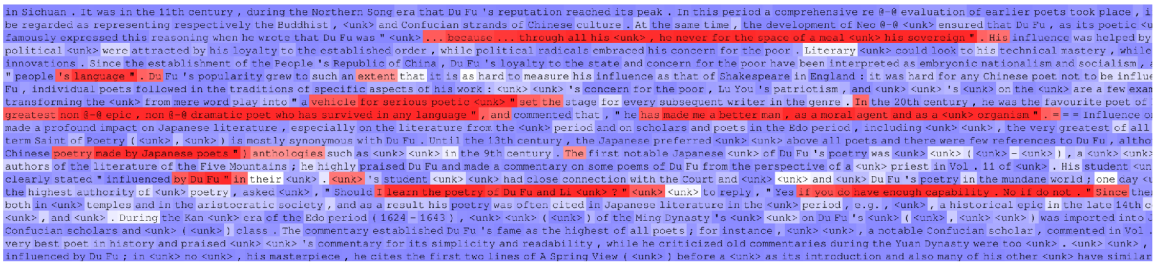
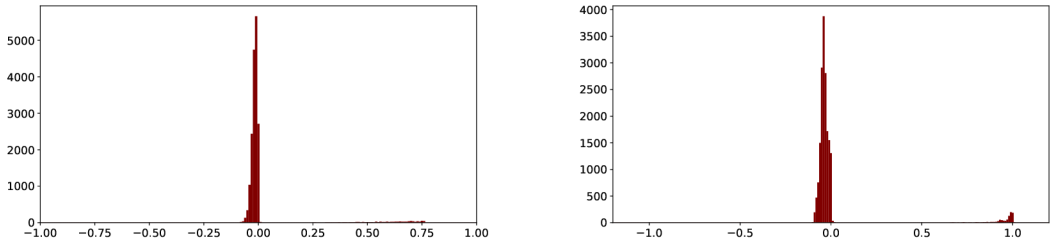


Figure 5.3: Blue and red graph where red shows high activations and is present only between quotation marks.

Because the values of neurons in the cell state are unlimited, it is harder to compare two probability distributions of neurons in the cell state. So to be able to compare neuron activation with the average neuron, I chose a fixed interval where values outside of the interval are not used. The interval size was chosen by observing what values the cell state contains. The neurons in the cell state exhibited very similar characteristics to the corresponding neurons in the hidden state. In Figure 5.4 is a neuron from the hidden state and from the cell state, and in both cases, the strong activation is present when the words on the input are between quotation marks. In other situations where a certain neuron predicts that the next token will be a number or <eos>, where <eos> is a pseudo-word meaning “end of sentence”.

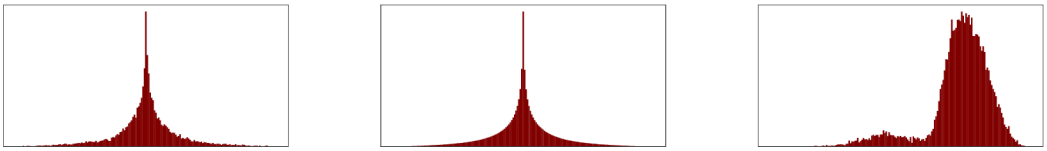


(a) Hidden state (b) Cell state

Figure 5.4: Activation of the same neuron in the hidden state and the cell state.

5.1.1 Filtering desired neurons

To filter these neurons, I used KL divergence. As a reference distribution, I used the average probability distribution of every neuron from the corresponding layer and state. Then I applied KL divergence, and because this method is a non-symmetric metric, I applied it from both directions. The neurons with the highest values are usually the ones that are interesting. The neuron with the lowest value is the one having a distribution almost matching the reference distribution. The comparison between neurons with high value and low value is shown in Figure 5.5 where the average is from the hidden state of the first layer.



(a) Min (b) Average (c) Max

Figure 5.5: Comparison of the differences between the neurons with high value and low value of KL divergence.

However, in the hidden state, the ones with the highest values are sometimes neurons that output mostly values very close to zero because the probability distribution of these neurons is also very different from the average distribution. Neurons like this have more than 90% values in the interval $<-0.01, 0.01>$. These neurons I did not analyze because I did not see any pattern in their activations. For instance, the neuron in Figure 5.6 is from the neural network with a dropout of 0.8 an and is analyzed on a test set. The analyzed text has 18 000 words, and the activations on over 17 700 words were in the interval $<0, 0.01>$ and over 150 were in the interval $<-0.01, 0>$.

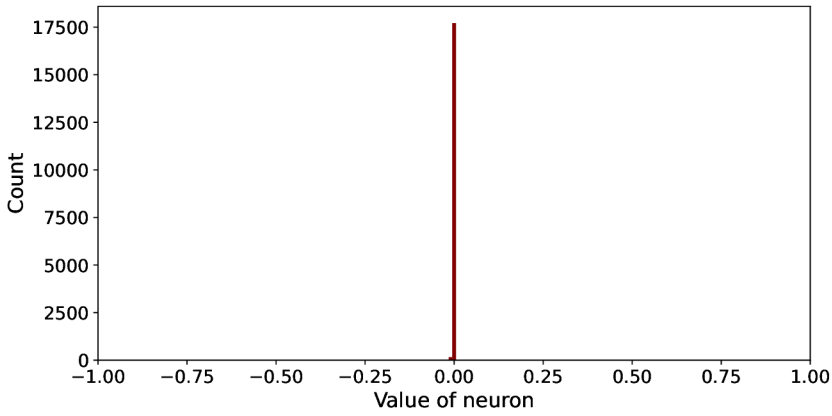


Figure 5.6: neuron with values mostly around 0.

5.1.2 occurence based on dropout

I compared the mean of the absolute value of activation on six models with different dropouts. Figure 5.7 shows that the higher the dropout of the model, the lower the mean. This characteristic is manifested in the model with a dropout of 0.8 by the fact that the model contained a lot of neurons that behaved in the same way as the neuron from Figure 5.6.

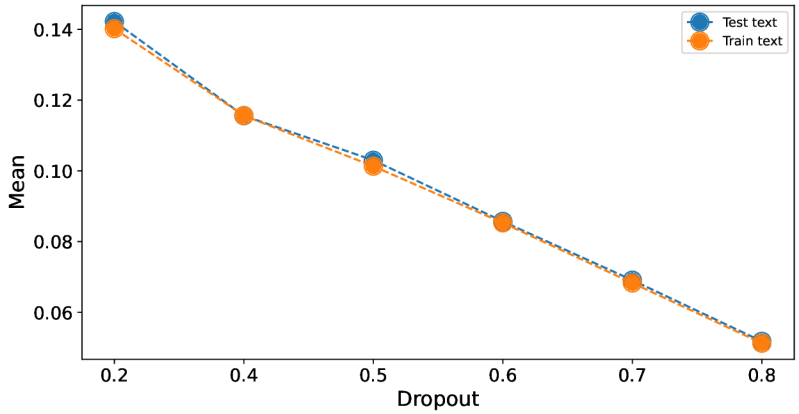
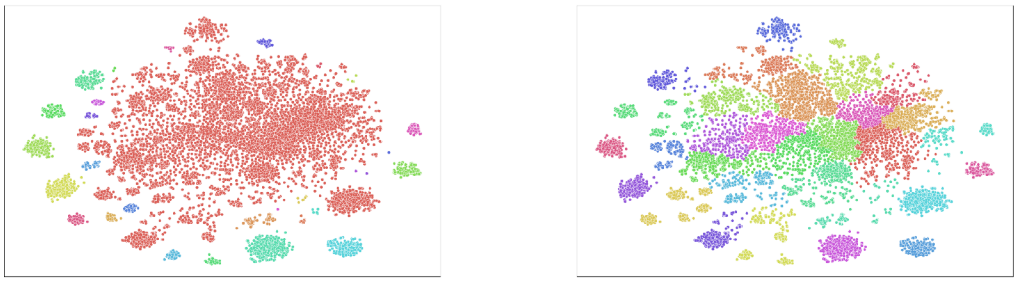


Figure 5.7: The mean of the absolute value of activation based on the dropout of the model.

5.2 Representation of a word

I represent words by activations of neurons when a certain word is processed. The text used to create the visualization is from training data and test data of Wikitext-2. Both texts had the most frequent work definitive article “the”, and it blended into a large part of the clusters. So the word is represented by 2000 values based on the layer and state. The expectation was that words with similar meanings would be close to each other. But to ease the clustering process, I converted the point representing the word from 2000-dimensional space to two-dimensional using t-SNE algorithm. For clustering, I used the Agglomerative

hierarchical clustering method. Because of the shape of t-SNE output, the ward linkage was not really useful because it divided the main central cluster into smaller pieces and was not connecting the clusters it should, as shown in Figure 5.8.



(a) Single linkage (b) Ward linkage

Figure 5.8: Comparison between ward linkage and single linkage on the hidden state in the first layer.

The downside of a single linkage is that it often creates clusters with a small number of points.

5.2.1 First layer

In the first layer, both the hidden state and the cell state have one central cluster and many separate smaller clusters surrounding them. For instance, in Figure 5.9 is a visualization of the hidden state in the first layer of the neural network with a dropout 0.7 on training text.

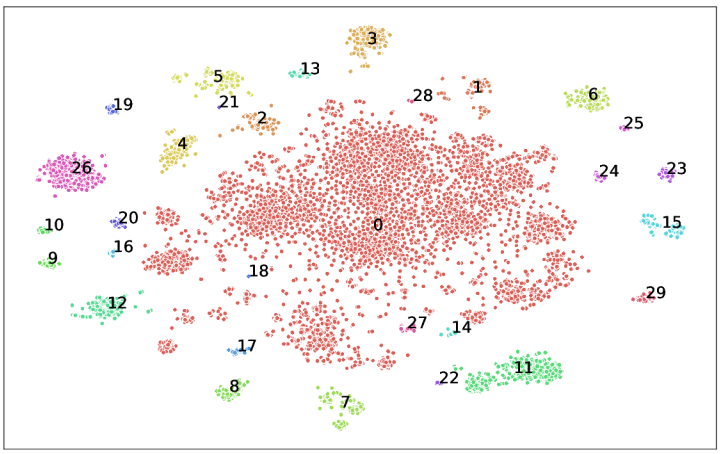


Figure 5.9: Clusters on training data in the first layer in hidden state.

The values contained in clusters in this layer have the largest amount of dependencies, so I represented values of the selected cluster in Table 5.1 and in the left column are

0	the, <unk>, of, with, The, for, that, @.@, season, Cullen
1	she, he, He, it, She, they, It, who, They
3	and, but, or, so, nor
5	was, is, were, be, being, are, been, became, remained, become
7	's, his, her, their, ', its, Her, His, elder, Its
12	in, In, during, into, During, throughout, By
15	=
19	had, has, have, having

Table 5.1: Table of values in clusters.

words from the cluster[2] arranged in descending order. In this table the numbers in the left column represent a number of a cluster from Figure 5.9. The cluster with the most values is labeled by the number 0. This cluster consists of a large number of different words but also contains article "the". Even though the article "the" is the most common word in the text, all of them are contained in this cluster. This situation arose because the cluster of the article is blended among the other words of cluster 0. This observation also applies to the article "The" and the pseudo-word "<unk>". Then in other clusters, we can observe a group of words that have similar semantics. The words were usually divided into groups of pronouns, conjunctions, indefinite articles "a", indefinite articles "an", possessive pronouns, the verb "be" and its derivatives, and the verb "have" and its derivatives. Often there is a cluster containing only "=", which is a markup used in wikitext for titles. The punctuation marks were divided into groups of commas, semicolons, colons and dots. Left parentheses and right parentheses are in separate clusters for all models, but this was not always the case for quotation marks. As opposed to quotation marks opening parenthesis and closing parenthesis are different tokens, so it is easier to separate them in text. In models with lower dropouts, the opening quotation marks and the closing quotation mark were in the same clusters however, models with higher dropouts were able to separate them.

The behavior of the cell state in the first layer was very similar in terms of the shape of the cluster and the words that the clusters contain. These experiments were performed on training data. However, the results of the test data were fairly similar.

5.2.2 Second layer

In the second layer, both the cell state and hidden state have a different shape than in the first layer. In the hidden layer, the points are very close to each other, and by using a single linkage, almost everything was grouped into one cluster as shown in Figure 5.10. The majority of the surrounding cluster consists of only a few points with no shared meaning. However, every model managed to group units into one cluster. Otherwise, there are clusters containing word "have" and its derivatives, word "be" and its derivatives, and clusters only with months.

[2]If a number of different words in the cluster was too big then in the table are only the most common ones.

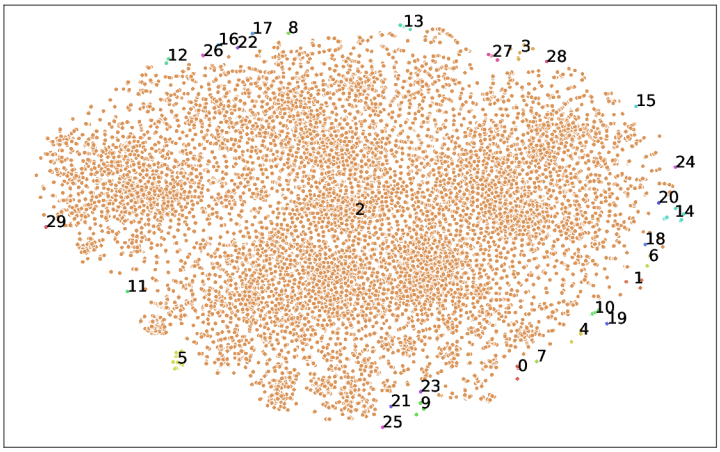


Figure 5.10: Hidden state second layer.

Because of the shape of the t-SNE output, I tried to use ward linkage and see if the words in these clusters have some similarities, but the clusters always have mixed values with the dominating word “the”. However, when I looked at the position of concrete words separately, they were not distributed completely randomly, as shown in Figure 5.11 with only the dot. They create a number of clusters, but they are separated.

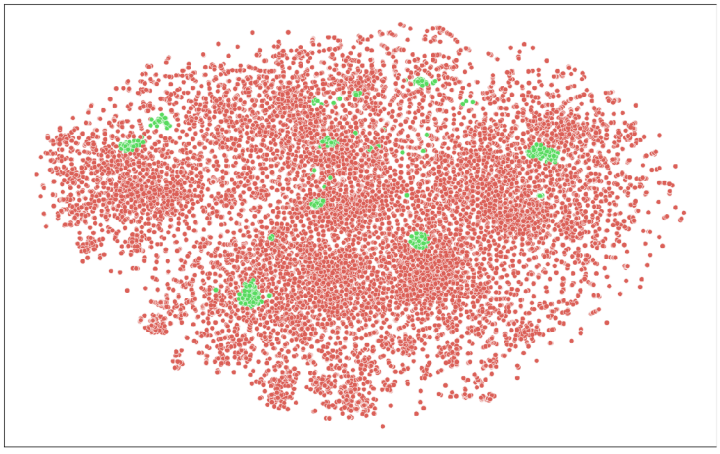


Figure 5.11: Green clusters represents dots.

In the cell state, the clusters often have elongated shape, which is conveniently clustered by using a single linkage. By listing all words contained in the cluster, I found no similarities. This was expected because the role of the cell state is not to connect words with similar meanings but to store the context. With that in mind, I looked at the position of the words in the analysed text. Turns out that the clusters contain consecutive words that

represent a section of a text with a certain context. For instance, in the Figure 5.12 a cluster labeled by the number 5 contains a sequence of words creating a connected part of text with the context of a hockey season. In cluster 13 are words representing a sequence smoothly following the sequence present in cluster 5 and ends exactly when the section talking about a hockey season is finished.

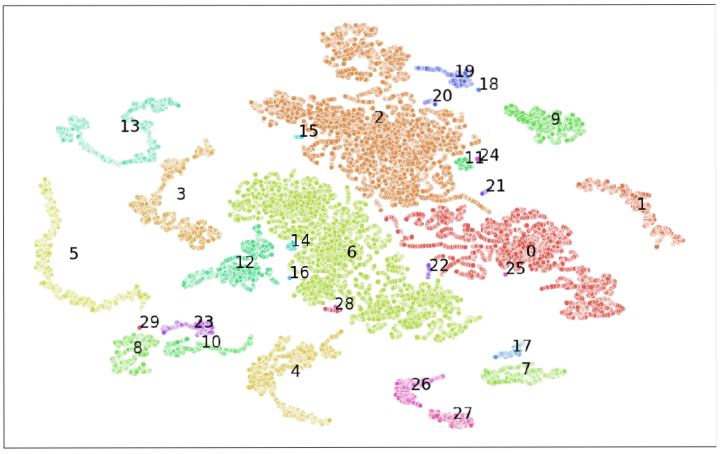


Figure 5.12: Shape of the clusters of cell state in the second layer.

Chapter 6

Conclusion

In this thesis, I presented three visualization techniques to provide information about the behavior of LSTM models. One technique uses t-SNE algorithm to visualize a high-dimensional representation of the words by the LSTM hidden and cell states values. From experiments conducted with this technique, I found that in the first layer, models are able to put together a group of words based on syntactic and semantic meaning. This behavior has been observed in both the hidden state and cell state. In the second layer, the hidden state, the individual words were divided into multiple smaller clusters. However, those words were usually mixed into one dominant cluster, and in rare cases, when the cluster was outside of the dominant cluster, similar behavior to the first layer was observed. The cell state in the second layer exhibits different properties, which are derived from the role of the cell state in LSTMs. Clusters contain a sequence of consecutive words which were referring to the same context.

The next visualization is focused on the unique patterns in activations of a single neuron. These activations are converted into a probability distribution, and by that, I managed to filter neurons that are different from the average. Among filtered neurons are those with uncommon activation patterns. The last method stores values present in the model when processing the individual word and convert this value into a color where the red color indicates high and the blue color indicates low. This method is suitable for visualizing what is behind the pattern of the activation of the filtered neuron as well as for visualizing the expectancy of the words by a model.

Bibliography

[1] Bengio, Y., Ducharme, R., Vincent, P. and Janvin, C. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* JMLR.org. march 2003, vol. 3, p. 1137–1155. ISSN 1532-4435.

[2] Chen, S. F. and Goodman, J. An Empirical Study of Smoothing Techniques for Language Modeling. In: *34th Annual Meeting of the Association for Computational Linguistics*. Santa Cruz, California, USA: Association for Computational Linguistics, June 1996, p. 310–318.

[3] He, K., Zhang, X., Ren, S. and Sun, J. *Deep Residual Learning for Image Recognition*. 2015.

[4] Hinton, G. E. and Roweis, S. Stochastic Neighbor Embedding. In: Becker, S., Thrun, S. and Obermayer, K., ed. *Advances in Neural Information Processing Systems*. MIT Press, 2002, vol. 15.

[5] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*. 1997, vol. 9, no. 8, p. 1735–1780.

[6] Maaten, L. van der and Hinton, G. Viualizing data using t-SNE. *Journal of Machine Learning Research*. november 2008, vol. 9, p. 2579–2605.

[7] Mikolov, T., Chen, K., Corrado, G. and Dean, J. *Efficient Estimation of Word Representations in Vector Space*. 2013.

[8] Mikolov, T. *STATISTICAL LANGUAGE MODELS BASED ON NEURAL NETWORKS*. Brno, CZ, 2012. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: `https://www.fit.vut.cz/study/phd-thesis/283/`.

[9] Olah, C. *Understanding LSTM Networks* [online]. August 2015 [cit. 2023-11-04]. Available at: `https://colah.github.io/posts/2015-08-Understanding-LSTMs`.

[10] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*. 2014, vol. 15, no. 1, p. 1929–1958.

[11] Zaremba, W., Sutskever, I. and Vinyals, O. *Recurrent Neural Network Regularization*. 2015.