VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
INFORMATIKOS KATEDRA

Effectiveness of pre-trained morphological parsers for text classification

Iš anksto apmokytų morfologinių analizatorių efektyvumas teksto klasifikacijai

Master Thesis

Author: Vytenis Šliogeris (signature)

Supervisor: Prof. dr. Olga Kurasova (signature)

Reviewer: Dr. Andrius Vytautas Misiukas Misiūnas (signature)

Vilnius – 2024

Summary

State-of-the-art models for natural language processing (NLP), such as transformers, employ positional encodings, which, according to our hypothesis, are not as effective at extracting grammatical functions of words for highly inflected languages. This is due to highly inflected languages employing a free word order, making the relative positions of words less related to the meaning of the words, unlike in non-inflected languages. A submodule for extracting grammatical information has been constructed and pre-trained in order to aid the overall model in natural language processing tasks for highly inflected languages. Text classification tasks were performed in the Lithuanian language to evaluate the effectiveness of such a submodule. It was found that the submodule did not significantly improve the performance in solving text classification tasks, with suggestion that such a submodule may be detrimental, yet it could make the learning process more stable.

Keywords: artificial intelligence, machine learning, natural language processing, morphological analyser, text classification

Santrauka

Šiuolaikiniai skaitmeninės natūraliosios kalbos apdorojimo modeliai, tokie kaip transformerių neuroniniai tinklai, naudoja padėties kodavimą, kuris, anot mūsų hipotezės, negali veiksmingai išgauti linksniuojamų žodžių gramatinių funkcijų. Skirtingai nei nelinksniuotose kalbose, linksniuotų kalbų žodžių pozicijos sakiniuose turi mažą svarbą nustatant minėtų žodžių gramatinę funkciją. Darbe sukurtas ir iš anksto apmokytas modulis, skirtas gramatinės informacijos išgavimui, kad modeliui būtų lengviau suprasti žodžių funkcijas sakiniuose. Atliktos teksto klasifikavimo užduotys lietuvių kalba, siekiant įvertinti tokio modulio efektyvumą. Nustatyta, kad gramatinės informacijos modulis nebuvo reikšmingas siekiant pagerinti teksto klasifikavimo uždavinių sprendimo našumą. Manoma, kad toks modulis gali būti žalingas, nors jis galėtų stabilizuoti mokymosi procesą.

Raktiniai žodžiai: dirbtinis intelektas, mašininis mokymas, skaitmeninis natūraliosios kalbos apdorojimas, morfologinis analizavimas, teksto klasifikacija

Contents

1. Introduction 4
   1.1. Justification for research 4
   1.2. Research object 5
   1.3. Research hypothesis 5
   1.4. Target 5
   1.5. Expected results 6
2. Literature review 7
   2.1. Statistical methods for NLP 7
   2.2. Artificial neural networks for NLP 9
   2.3. Attention mechanism 13
   2.4. The transformer architecture 15
   2.5. NLP tasks 17
   2.6. Word representation 18
   2.7. Data augmentation in NLP 22
   2.8. NLP in inflected languages 24
   2.9. Literature summary 27
3. Methods and results 29
   3.1. Datasets 29
      3.1.1. Morphologically annotated corpus 29
      3.1.2. Parliamentary speech data 29
      3.1.3. Financial sentiment statements 32
   3.2. Model construction 33
      3.2.1. Grammatical case model 33
      3.2.2. Task models 34
      3.2.3. Parliamentary speech classification model 36
      3.2.4. LSTM for financial sentiment analysis 37
   3.3. Training 38
      3.3.1. Grammatical case model training 38
      3.3.2. Parliamentary speech classifier training 38
      3.3.3. Financial sentiment classifier training 42
   3.4. Model analysis 44
      3.4.1. Grammatical case model results 44
      3.4.2. Parliamentary speech classifier results 45
      3.4.3. Financial sentiment analysis results 48
4. Conclusion 50
   4.1. Future work 50

References 52

1. Introduction

1.1. Justification for research

Grammatical case describes the grammatical function of nouns [Fre94]. Languages such as English have mostly lost grammatical cases and only retain inflections in personal pronouns. Since a grammatical case encodes the function of a noun in a sentence, and hence some information, English retains such information by the position of the word in the sentence. Languages that use grammatical cases tend to allow a free word order, as the functions of nouns are encoded in the words themselves.

Models such as the transformer [VSP+17] include positional encoding in word embeddings so that the neural network can better use the crucial positional information. However, in languages that have retained their grammatical case, positional encoding is not as informative, as the word order is less strict. Unlike positional encodings, there are no dedicated parts of the model for encoding word functions for highly inflected languages. It could be the case that noun cases may be learned in the word embeddings, with two nouns that only differ by the grammatical case having similar vector representations.

Morphological parsing is a popular step in natural language processing (NLP) for various highly inflective languages, such as Turkish [Ery14], Tamil [JRR+11], Arabic [AEY+08], Lithuanian [RDU07], and Czech [Hla00]. They can take the form of rule-based parsers or hidden Markov models and aim to assign morphological features to words. To the author's knowledge, this morphological information has not been used to aid in classification tasks.

Text classification is the problem of assigning tags from a predetermined set to bodies of text [CFK+20]. Multi-label text classification is considered to be a more difficult case of text classification, where each body of text may have multiple labels assigned to it, which may skew the distribution of labels, introducing biases. Great relevance has been placed upon text classification, making it an actively researched subject. It has been used for tagging clinical notes with the appropriate International Classification of Diseases (ICD) codes [MWD+18]. ICD codes can be used for patient billing and modelling patient states. This process is labour intensive and prone to errors; hence, it has been researched since the 1990s. The legal domain has also seen the use of text classification, assigning legal concepts [CFM+19] or legal judgement [ATP+16] to documents. This can aid in explaining legal risk, provide insights for lawyers, determine biases [PD05] [Che19], predict court decisions [LAS+22] [KBB17] [MVW20] and test legal theories. A similar classification was performed on datasets of the Lithuanian language. Single-label classification of Lithuanian Parliament legal texts [MKM15] has been performed, which aimed to assign topics to parliamentary briefings.

The purpose of this work is to examine whether the inclusion of morphological information in models for highly inflected languages aids in NLP tasks. The task of text classification was chosen, because it is a highly researched topic, thus there are many existing problems, and advances in such problems can aid in other domains.

1.2. Research object

In this work, several models for text classification can be considered across a number of datasets in the Lithuanian language, such as the LSTM, BERT [DCL+18] and transformer [VSP+17]. The construction of a model using a morphologically annotated corpus [RBD+19], which can extract grammatical noun cases from Lithuanian words, will be attempted. The effectiveness of a model performing text classification tasks with and without grammatical cases will be examined. Data pre-processing will be investigated to help remedy the inherent skewness of the data with regard to targets, which leads to overfitting.

1.3. Research hypothesis

It can be hypothesised that including a submodel that extracts the morphological information of words and appends it to word embeddings may increase model accuracy or decrease training time. This may be due to the classifier network not needing to extract the morphological features from the words during training, as this functionality would be done by a separate submodel, which would train on data particularly catered for this task. Unlike positional information, such a submodel would not include new, essential information to the model, therefore accuracy gains are not guaranteed.

1.4. Target

The goal of this work is to determine the extent to which including morphological information in various text classifiers changes the results for highly inflected languages, such as Lithuanian. A submodel for extracting the grammatical function of nouns will be constructed and evaluated. It would perform a similar function to the positional encodings in the transformer.

To achieve this goal, the following objectives will be addressed:

1. A literature review will be performed. Papers regarding annotated corpora in the Lithuanian language will be studied. Existing examples of NLP tasks in the Lithuanian language will be read upon. Surveys of potential tasks for evaluating models will be reviewed.

2. Construction of a submodel which can determine the grammatical case of words will be attempted with the aid of a morphologically annotated corpus. It would take form of a pre-trained classifier, classifying words into their respective noun cases.

3. A model enhanced with the grammatical case submodel will be evaluated and compared with the regular version of the model in previously researched tasks. Changes in the number of training epochs and changes in accuracy will be compared between the models as the metric for evaluation.

An experiment comparing the results of existing models with those of enhanced models will be performed. The models considered will be LSTM, BERT [DCL+18], transformer [VSP+17], and any additional models discovered during the literature review.

These models will be used on Lithuanian datasets. The Lithuanian datasets used will include the historical roll call votes of the Lithuanian Parliament, accessible in its website [MKM15], and the Lithuanian financial news dataset [ŠSR+21], accessible on Kaggle.

1.5. Expected results

The hypothesis is that a model augmented with the case submodel will produce greater accuracy than the original models. It is also expected that the accuracy will increase faster for the augmented models than for the original models.

2. Literature review

2.1. Statistical methods for NLP

Natural language processing has been considered for as long as modern digital computers have existed [BCD+90]. Natural language processing has gone through several phases in its history. Currently, NLP is based on machine learning and neural networks. In the previous era of NLP, statistical methods were utilised. One of the earliest papers introducing such methods is [BCD+90] for the purpose of translating French text to English. Statistical methods aim to build statistical models and learn the underlying probabilities to try to accurately produce desired behaviour and outputs. In this case, the process of building the model is two-fold. First, the parameters in the form of probabilities are estimated using a large language corpus. They are used to estimate the probabilities $P(T|S)$, where S is a sentence in the source language, T is a sentence in the target language, and $P(T|S)$ is the probability that sentence S translates to T. However, the number of possible arguments for T is very large, therefore a search algorithm needs to be utilised.

There are some recurring ideas in this paper that have lasted until these days. For example, they describe the idea of predicting the following word given a string of previous words. Namely, they describe that one can decompose a word string $s_1, s_2, \ldots s_n$ into individual probabilities as such:

$$P(s_1 s_2 \ldots s_n) = P(s_1)P(s_2|s_1)\ldots P(s_n|s_1 s_2 \ldots s_{n-1}). \tag{1}$$

This type of reasoning is ubiquitous in modern language models, as in addition to considering the input sentence in the source language they also consider the sentence built so far.

The concept of word alignment has also been highlighted in this work. A word in the source sentence and in the target sentence is considered aligned if they appear in the same positions in their respective sentences. It is clear that words in source and target language often do not align, and notation for the corresponding word position has been developed by the authors, which played a part in the construction of the translation model. The concept of alignment reemerged in 2014 in the form of the attention mechanism, which became a breakthrough in natural language processing.

This area of research culminated in machine translation systems such as Moses [HK08] in 2008, which became the „One-Stop Shop" for Phrase-Based statistical machine translation. However, these models had several drawbacks, as described by [KB13]. Similar words and phrases may be semantically or statistically similar, but not logically connected in the models. This means that numerous parameters have to be learned for separate phrases that essentially mean the same thing. For that similar reason, there is a large sparsity issue, where only a few phrases containing a particular word are present in the dataset and are learned by the model. This makes the models quite domain-limited. The complexity of such a model would substantially increase when trying to generalize it.

An evaluation metric known as BLEU (Bilingual Evaluation Understudy) was introduced by [PRW+02] and has been used to evaluate and compare machine translation models. It stems from a language-independent evaluation method for automatic machine translation, which correlates

highly with human evaluation. It compares a produced translated sentence with several reference statements. The aim is to evaluate how close the produced sentence is to the reference statements. It is a weighted geometric mean of multiple different n-gram precision scores, multiplied by a brevity penalty factor.

The n-gram precision scores are computed by counting up the number of words in the candidate sentence that occur in any reference sentence and then dividing by the length of the sentence. One drawback of this approach is that a sentence containing a single word (or multiples of a single word) may score unreasonably high. A sentence, such as *the the the the the the the* may score very well if one of the reference sentences has a word *the* included. Therefore, the precision metrics were modified to „consume" the matching word. In the previous example, if the reference sentence has only one instance of the word „*the*", then only one of the *the*s will match, giving a very low score.

A brevity penalty is also introduced to enforce a similar sentence length, which is found in the reference sentences. A subsentence of a reference sentence would make a very good candidate, since the word in such a candidate would entirely show up in the references. These problems would be overcome with a recall metric, however, since there are multiple reference sentences, the resulting behaviour would prefer mashups of the reference sentences rather than being close to one of the references. Thus, a brevity penalty penalizes sentences that are shorter than the references. There is no need to penalize sentences which are too long, as they are already penalised by the n-gram precision. The reference length to compare with is computed over the whole sentence corpus. The value c is computed by summing all the sentence lengths of the candidates of the entire corpus, and the value r is computed by summing the best matching reference lengths for each candidate sentence in the corpus. The brevity penalty is computed via Eq. (2).

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-\frac{r}{c}} & \text{otherwise.} \end{cases} \tag{2}$$

With the brevity penalty in hand, the actual BLEU value is computed using Eq. (3).

$$BLEU = BP \cdot e^{\sum_{n=1}^{N} w_n \log p_n}. \tag{3}$$

The metric ranges from 0 to 1. p_n is the modified n-gram precision score, and w_n is the weight for that particular metric. The exponential form of the equation was chosen rather than a linear combination of p_n since they experimentally found that the n-gram precision exponentially decreases as n increases. The exponential form of the equation takes the aforementioned finding into account.

It has been found to correlate well with human judgement. [PRW+02] compared BLEU to human evaluators. The authors got native English speakers and native Chinese speakers with knowledge of English to rate translations of sentences from 1 to 5. The human evaluations correlated with the BLEU scores with a correlation coefficient of 0.96. For this reason BLEU has been a very popular evaluation metric, especially among statistical methods.

2.2. Artificial neural networks for NLP

A revolution occurred regarding NLP in 2010 with the introduction of artificial neural networks (ANNs) [MKB+10] in the form of recurrent continuous translation models [KB13]. These models were based on recurrent neural networks (RNN). According to [SMH11], RNNs transform a sequence of input vectors $(x_1, \ldots, x_T)$ into a sequence of hidden states $(h_1, \ldots, h_T)$ and a sequence of outputs $(o_1, \ldots, o_T)$. It is performed by iterating over Eqs. (4) and (5).

$$h_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{4}$$

$$o_t = W_{oh}h_t + b_o. \tag{5}$$

W_{hx} is the input-to-hidden weight matrix, W_{hh} is the hidden-to-hidden weight matrix, and W_{oh} is the hidden-to-output weight matrix, while b_h and b_o are the biases for the hidden state and the output state, respectively. These are the parameters learned during training. The RNN modifies its hidden state according to each input, while returning an output at each step. The model acts as a single neuron layer that continuously updates some hidden state.

They were introduced in conjunction with continuous word representations, currently known as word embeddings, which will be discussed in more detail later in this work. These models were empowered by word embeddings, since neural network-based models work best with real valued vectors. While word embeddings captured the meaning and similarity between words, ANNs were shown to be able to capture semantic and syntactic information and soon became a preferable model type.

RNNs are not without flaws, as they are prone to exponentially decaying gradients. It comes from the reasoning presented by [HS97], which will be briefly described. The gradient instability stems from the definitions of backpropagation. For an output node k at time t, the mean squared error for a single datapoint of the RNN will be such:

$$e_k(t) = (d^k(t) - y^k(t))^2, \tag{6}$$

where $d_n^k(t)$ is the target for the neuron, and $y_n^k(t)$ is the value produced via Eq. (7).

$$y^i(t) = f_i(net_i(t)). \tag{7}$$

f_i is the activation function for node i, and net_i denotes the weighted sum of the input nodes.

$$net_i(t) = \sum_{j \in J} w_{ij} y^j(t-1), \tag{8}$$

where J denotes the set of input nodes for i, w_{ij} is the weight from node j to node i, and $y^j(t-1)$ denotes the values of the input nodes at the previous timestep. This is exactly how a linear perceptron works. For an output node, the error $\theta_k(t) = \frac{\delta e_k(t)}{\delta w_{uv}}$ propagated back will be the derivative

of that value with respect to a particular weight, described as follows:

$$\theta_k(t) = f_k^{'}(net_k(t))(d_k(t) - y^k(t)). \tag{9}$$

$f_k^{'}$ is the derivative of f_k and appears when deriving $y^k(t)$. For any other node at a non output timestep, the error propagated back $\theta_j(t)$ takes the following form:

$$\theta_j(t) = f_j^{'}(net_j(t))(\sum_{i \in I} w_{ji} y^i(t+1)). \tag{10}$$

This signal is then used to adjust a weight w_{ji} via a learning rate α.

$$w_{ji} := w_{ji} + \alpha \theta_j(t) y^i(t-1). \tag{11}$$

As per [Hoc91], the change of the error signal at node u at a time q steps ago with respect to the current error of node u is computed. There are two separate cases that produce a recursive function. The first case is $q = 1$, i.e. the error goes back one timestep.

$$\frac{\delta \theta_v(t-1)}{\delta \theta_u(t)} = f_v^{'}(net_v(t-1)) w_{uv}. \tag{12}$$

This equation is obtained by deriving Eq. (10) with respect to $\theta_u(t+1)$. This can be considered the base case of the recursive equation. For $q > 1$, the equation takes the following form.

$$\frac{\delta \theta_v(t-q)}{\delta \theta_u(t)} = f_v^{'}(net_v(t-q)) \sum_{l=1}^{n} \frac{\delta \theta_l(t-q+1)}{\delta \theta_u(t)} w_{lv}. \tag{13}$$

This equation is produced by applying the derivative to Eq. (10), where the derivative inside the sum does not simplify.

The recursive expression can be computed via induction as follows:

$$\frac{\delta \theta_v(t-q)}{\delta \theta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f_{l_m}^{'}(net_{l_m}(t-m)) w_{l_m l_{m-1}}. \tag{14}$$

Intuitively, if the inner product $f_{l_m}^{'}(net_{l_m}(t-m))w_{l_m l_{m-1}}$ is greater than one, the error at further timesteps back increases exponentially as q increases, making training incredibly unstable. In contrast, if that value is less than one, that value exponentially decreases as q increases, making the model unable to train.

To counteract these exploding and vanishing gradients, a constant error signal would be preferable. The error flow following a single timestep along a single node can be derived from Eq. (10), producing in the following equation:

$$\theta_j(t) = f_j^{'}(net_j(t)) \theta_j(t+1) w_{jj}. \tag{15}$$

Since we want the error to be the same $\theta_j(t) = \theta_j(t+1)$, the necessary condition is as follows:

$$1 = f_j^{'}(net_j(t))w_{jj}. \tag{16}$$

Integrating the equation we obtain the following equation:

$$\frac{net_j(t)}{w_{jj}} = f_j(net_j(t)). \tag{17}$$

The uncomfortable implication is that implementing this naively would mean that the activation function f has to be constant and that the weight w_{jj} would have to be 1.

LSTMs were introduced by [HS97] to address decaying or exploding gradients that occur when using RNNs. It is an extension of the RNN, where an additional memory cell is introduced with gates that change the contents of the cell in a predictable way. These gates would introduce more parameters that would allow the model to deactivate some weights depending on input, addressing some issues outlined in the previous paragraph. The proposed original LSTMs had two gates: the input gate and the output gate.

An input gate acts as a gate keeper which determines how much new input should change the memory. It influences the cell state by determining whether the received input should change the state. It is done by computing an input gate's activation vector, Eq. (18).

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i). \tag{18}$$

It takes the input x_t, the hidden state previously received h_{t-1}, and passes them through perceptrons denoted via weight matrices W_i and U_i. σ_g denotes a sigmoid function. This results in a vector with values between 0 and 1.

This changes the input via Eq. (19).

$$c_t = c_{t-1} + i_t \odot \tilde{c}. \tag{19}$$

$\odot$ refers to point-wise product, and $\tilde{c}$ is the actual change to be applied. These are computed via Eq. (20).

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c), \tag{20}$$

which acts in the same way as the input gate in Eq. (18), with the sigmoid function replaced by a hyperbolic tangent function σ_c, resulting in the vector being in the range $(-1; 1)$.

An output gate works similarly to an input gate, except it determines whether the output should be changed. The gate takes the form of Eq. (21).

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o). \tag{21}$$

This vector with values between 0 and 1 determines how much of the memory cell is present in the hidden state via Eq. (22).

$$h_t = o_t \odot \sigma_c(c_t). \tag{22}$$

There exist other variations of the LSTM, such as the LSTM with a forget gate, Peephole LSTM, etc. The forget gate is a popular addition, which modifies whether or not the cell data gets overridden. This makes it behave less like an RNN. The forget gate, similar as the input and output gate, takes form as Eq. (23).

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f). \tag{23}$$

It is similar to the input gate Eq. (18) and the output gate Eq. (21). It produces a similar vector, and it changes the state according to a modified form of Eq. (19), seen in Eq. (24).

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}. \tag{24}$$

A projected version of the LSTM architecture, the LSTMP, has been proposed by [SSB14]. It attaches a linear projection layer after the LSTM. The idea is to increase the dimensionality of the LSTM memory cell and, when needed, project it to a lower dimension. This allows us to increase the size of the memory and to control the number of parameters. These changes modify the equations a bit. A projection layer is introduced in Eq. (25).

$$r_t = W_{rh} h_{t-1}. \tag{25}$$

The output in Eq. (22) is changed to use this projected value instead, where ϕ denotes the activation function, for example, softmax.

$$h_t = \phi(W_{yr} r_t + b_y). \tag{26}$$

In order to further reduce the number of parameters needed in an LSTMP, [KG17] have utilised factorisation (F-LSTM). They first consider the computation of the gates as matrix multiplications.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} T \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}. \tag{27}$$

x_t is the input, h_{t-1} is the previous state of the LSTM, and $sigm$ with $tanh$ are the notations for the sigmoid and the hyperbolic tangent functions, respectively, as used by the authors. T is an affine transformation that converts a vector of length $2p$ to a vector of length $4n$.

$$T = W \cdot [x_t, h_{t-1}] + b. \tag{28}$$

W is the matrix to be trained for this affine transformation, which represents all the gates in the LSTM. The shape of the matrix must be $4n$ by $2p$ so that the dimensions match on both sides of the equation.

$$W = W_2 \cdot W_1. \tag{29}$$

The key insights are that W may be approximated by two smaller matrices, $W2$ and $W1$. The shapes of these matrices must be r by $4n$ and $2p$ by r, respectively. These two matrices together have fewer parameters ($r(2p + 4n)$) than the original matrix ($8np$).

Gated recurrent units (GRUs) have been proposed by [CVG+14] as a more lightweight version of the LSTM. It has been constructed upon RNNs and inspired by LSTMs, where the RNN has a reset and an update gate. These equations are analogous to LSTMs (e.g. Eq. (23)), with these gates being remembered and learned via matrices. The essential part in their training is how they are applied, and the gates are used via Eqs. (30).

$$\begin{aligned} c_t &= zc_{t-1} + (1 - z)\hat{c}_t \\ \hat{c}_t &= \phi(Wx + U(r \odot c_{t-1})). \end{aligned} \tag{30}$$

The reset gate r determines which values to ignore from the previous hidden state, and the update gate z determines how much of the new state will override the old one. Multiple of these units may be used to learn time-dependencies over multiple time spans. This model, while being significantly more compact than the LSTM, has similar performance.

BiRNNs and BiLSTMs are popular extensions of RNNs and LSTMs. They address the drawback that these models operate in one direction, generally reading tokens from the start to the end of sequence. The final token will generally have the strongest effect in the sentence, since its resulting state will be overridden by the fewest number of following tokens, while the first token in the sequence may be entirely forgotten. To balance this asymmetry, it has been proposed to run two separate models forwards and backwards. The forward model is generally $\overrightarrow{RNN}$ and backward model is $\overleftarrow{RNN}$, with the arrow direction corresponding to the direction along which the model reads tokens. Both models share the same architecture but have separate weights. They produce a state as such:

$$\begin{aligned} \overrightarrow{h} &= \overrightarrow{RNN}(x) \\ \overleftarrow{h} &= \overleftarrow{RNN}(x). \end{aligned} \tag{31}$$

To avoid losing any information, these two states are usually concatenated and used as a single state for downstream architectures.

$$h = [\overrightarrow{h}, \overleftarrow{h}] \tag{32}$$

2.3. Attention mechanism

The addition of the attention mechanism has been an addition that has been persistent in most successful iterations of models that reach state-of-the-art results. First introduced in 2014 [BCB14], it facilitated the establishment of connections between directly translated words, which often have different positions in their respective sentences.

In practice, [BCB14] transformed the encoded state of the input sentence into a context vector using the Eq. (33), which returns a single vector as a weighted sum of all input word embedding vectors, giving most weight to the most important words.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{33}$$

The weights α_{ij} are computed using Eq. (34).

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}. \tag{34}$$

e_{ij} is the value given by the alignment model via Eq. (35).

$$e_{ij} = a(s_{i-1}, h_j). \tag{35}$$

The alignment model scores how well the words at input position j and output position i match. It is implemented as a feedforward neural network, which is trained alongside the entire system.

This connection essentially forces the decoder to focus on a corresponding word in the input sequence. In statistical models, this was implemented using the concept of alignment, saving the resulting word position with the tokens as integers.

Another influential attention model is the self-attention model. Self attention signifies weighting a sequence of tokens using the information within that same sequence of tokens. Self attention on bidirectional LSTMs has been implemented by [LFS+17] to produce sentence embeddings which weigh more important tokens more heavily. The attention model works on the hidden state H, described in Eq. (36).

$$H = (h_1, h_2, \ldots, h_n). \tag{36}$$

The hidden state is a two-dimensional matrix of shape n by $2u$, where n is the number of tokens of a sentence, and u is the size of the hidden state of the LSTM. One dimension of the matrix is $2u$ because the output of the forward and the backward LSTM are concatenated. The self-attention vector a is computed via Eq. (37).

$$a = \sigma(w_{s2} \tanh(W_{s1} H^T)). \tag{37}$$

W_{s1} is a weight matrix of shape d_a by $2u$, which is a learned parameter. d_a is a hyperparameter, chosen by the researcher. The product of $W_{s1} H^T$ results in a matrix of shape d_a by n. w_{s2} is another learned vector of length d_a, so that the result of the previous computation results in a vector of length n. The softmax function σ establishes that a sums up to 1, and thus the values of a signify the importance of each token in the sentence. The sentence embedding m can thus be computed via dot product $m = aH$.

It may be necessary to learn multiple attentions, as there may be multiple important words in the sentence. Hence Eq. (37) is extended to produce Eq. (38).

$$A = \sigma(W_{s2} \tanh(W_{s1} H^T)). \tag{38}$$

The difference is that w_{s2} is replaced by W_{s2}, which is a matrix instead of a vector. The shape of that matrix is r by d_a, where r is a hyperparameter which signifies how much attention is necessary. Essentially, W_{s2} is multiple instances of w_{s2} stacked. The resulting embedded sentence becomes a matrix M of shape r by $2u$, computed via Eq. (39).

$$M = AH. \tag{39}$$

2.4. The transformer architecture

The seminal paper by [VSP+17] introduced the transformer architecture which utilises self-attention. It has spawned a wide range of successful models, such as BERT [DCL+18], ROBERTA [LOG+19], and GPT-2 [RWC+19], which is a predecessor to ChatGPT. A transformer is a set of feed-forward neural networks, which sacrifices the size of input tokens in favour of training speed, as recurrent models are difficult to parallelize. The model consists of an encoder and decoder stack. As in previous models, the input sentence is encoded to a vector representation $z = (z_1, \ldots, z_n)$, which is then decoded to a sequence in the target language $(y_1, \ldots, y_m)$. The encoder stack is made up of multiple identical layers, each of which is made up of two sublayers. The first sublayer is a multi-head self-attention mechanism, while the second one is a feed-forward network. The decoder is also a stack of layers, each of which is comprised of three layers. The first two are multi-head attention layers, while the third one is a feedforward layer as in the encoder. The first multi-head attention layer takes in the symbols decoded so far, while the second multi-head attention takes the result of the previous sublayer *and* the result of the encoder.

The two essential novelties presented were positional encodings and self-attention, which allowed the model to reach state-of-the-art results. The model employs scaled dot-product attention, of which there are multiple instances, which in totality are called multi-head attention. This attention takes the shape of Eq. (40).

$$Attention(Q, K, V) = \sigma(\frac{QK^T}{\sqrt{d_k}})V. \tag{40}$$

Q, K ,and V represent the query, key, and value, respectively, while d_k is the dimension of the model which is a scaling factor to avoid the result from exploding. Dot product self attention is preferred for its quick computation time. The input for this attention is the input word embeddings, each of which is projected using parameter matrices W^Q, W^K and W^V, which are the trainable parameters for the attention mechanism. There are multiple of these attention parameter matrices, which lets it learn multiple forms of attention, as shown in Eqs. (41) and (42).

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O. \tag{41}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V). \tag{42}$$

W_i^Q, W_i^Q and W_i^Q represent the parameters for the ith head for the query, keys and values. There is also an overall projection matrix W^O, which is applied to the concatenated results of all attention heads.

Since the model is not recurrent in nature, positional encodings encode the positions of the tokens. The encodings are computed as shown in Eqs. (43) and (44), where pos is the position, i is the dimension and d_{model} is the dimension of the word embeddings.

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}). \tag{43}$$

$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}). \tag{44}$$

Odd and even positions in the dimension of the encoding alternate between sine and cosine waves of increasing frequencies. The word embedding at position pos at dimension i gets added the value described in the equations. Since this encoding is the same for same words in the same position, the model can learn them and interpret them as representative of the position of the word.

An alternative to fixed positional encodings is learned positional encodings. A form of these is implemented by [GAG+17]. The absolute positions of tokens were embedded in vectors of real numbers.

$$p = (p_1, \ldots, p_m); p_j \in \mathbb{R}^f. \tag{45}$$

When [VSP+17] experimented with learned embeddings, they have found that the results achieved were nearly identical.

Transfer learning has become an important feature of contemporary language models. One of the first implementations has been produced by [DL15]. They have merged two approaches to pre-train LSTMs.

The first approach was that of an auto-encoder. An auto-encoder aims to compress data by learning the underlying structure of it. Generally, it is done using an encoder and a decoder network. An encoder network takes a sequence and produces a representation of the initial data which is smaller than the initial data. Then a decoder takes the representation and tries to reproduce the initial data. If this succeeds, then it can be stated that the data has been successfully compressed to and recovered from a smaller representation vector.

Another approach for pre-training is the use of a Language model (LM). A language model aims to predict a following token given a sequence of strings.

The synergy of the two models was produced by using an LSTM as an auto-encoder that functions as a LM. Instead of using two separate models, a single LSTM reads the entire sequence without producing any output. During this pass it encodes the input into its hidden state. Afterwards it receives the end of sequence token „<eos>", where it starts returning the original sequence, and then reads its own output to produce the following tokens. The compression is evident, as when the model reaches the end of the sequence of length n, it has to reproduce a token it has last seen n tokens ago.

Many of the offspring of the transformer lend themselves to transfer learning, where a model

is pre-trained with unsupervised learning, and is then available to fine-tune for a particular task, which contributed to its popularity. One such example is the bidirectional encoder representations from transformers (BERT) by [DCL+18]. BERT is a multi-layer bidirectional transformer encoder, based on [VSP+17]. The framework of training BERT is comprised of two steps - pre-training and fine-tuning.

The pre-training is performed on two unsupervised tasks - the Cloze task [Tay53] and next sentence prediction. For the Cloze task, some percentage of the input tokens are masked at random, and the model has to predict these masked tokens. During the next sentence prediction task, the model has to determine, given a pair of sentences, whether the succeeding sentence is next from the preceeding sentence in the dataset. This ask intends to train the model to gather an understanding of sentence relationships.

The architecture is notable for making the model extendable to many tasks. It is done by plugging task specific inputs and outputs into BERT and fine-tuning the parameters end-to-end.

2.5. NLP tasks

Question answering, also known as the machine reading comprehension (MCR) task, is a useful task for demonstrating the ability of machine learning models to understand natural language. The act of answering a question may be seen as multi-step reasoning, where given a question subject, the model has to gather what the subject is, and then connect the subject with the information demanded. An example would be the question „How many people live in the capital of France?", where the reasoning steps would be that Paris is the capital of France, and that Paris has a particular population.

Stochastic answer networks have been proposed by [LSD+17]. They define that the MCR task involves a question $Q = \{q_0, \ldots, q_{m-1}\}$ and a passage $p = \{p_0, \ldots, p_{n-1}\}$, with the goal of finding an answer $A = \{a_{\text{start}}, a_{\text{end}}\}$. It is assumed that the answer exists in the passage text. A solution for such a task would be a function $f(Q, P) = A$, which would learn from tuples $< Q, P, A >$.

The proposed model has 4 layers. The first layer is the Lexicon Encoding Layer, which processes the inputs Q and P to convert it to vectors. The vectors contain GLOVe [PSM14] embeddings, POS tags, named entity recognizer tags, etc. That results in each token in the passage being encoded as a 600-dimensional vector, and each token in the question being a 300-dimensional vector. The dimensions mismatch, since the passages get enhanced with an additional alignment embedding, which measures soft alignment between the word and any word in the question. The mismatching dimensions are problematic later on, and are mapped to a lower dimensionality d by feed-forward networks. The following layer is the contextual encoding layer, which converts the lexicon encodings to contextual embeddings. Bidirectional LSTMs are used to generate these contextual embeddings, akin to [PNI+18]. The sentences get passed through the LSTMs, and the states of the LSTM are used as the contextual embeddings. Since the whole sentence is examined, the same words may map differently depending on the sentence. Next is the Memory Generation Layer, which generates a memory vector by applying several instances of attention and another Bidirectional LSTM layer. Finally, the answer module generates the actual output. It keeps a state vector,

which evolves over each timestep according to the following equation, where s_{t-1} is the previous state and x_t is the computed memory, and GRU is the gated recurrent unit.

$$s_t = \text{GRU}(s_{t-1}, x_t). \tag{46}$$

The state vector is used to compute a beginning point P^{begin} and end point P^{end} via a linear layer. These points determine the words of the passage between which the answer to the question is. Multiple networks are trained to generate multiple estimates of these points, and an average of these points is used for the actual produced answer.

Textual entailment is a separate task for NLP to tackle. According to [CZL+16], the task is concerned with determining whether a hypothesis follows from a premise. A cornerstone corpus was the Stanford Natural Language Inference (SNLI) corpus, which provides pairs of statements and their relationship. A model for this task would take as input two sentences and predict whether or not there is a logical relationship.

Semantic role labelling aims to assign tokens in a sentence with a semantic role. [HLL+17] envision this problem as a tagging problem, where a sequence of labels is generated given a sequence of tokens. They utilise the BIO tagging scheme, introduced by [RM99], where each label can have one of three values $y_i \in \{O, B_r, I_r\}$. A tag O means that the word is unrelated to any arguments. B_r shows that the token is the beginning of role r, and I_r shows that it is inside I_r. The authors utilised deep BiLSTM models with recurrent dropout for this task.

2.6. Word representation

Data representation in NLP has also improved over the years. In the era of statistical methods, n-grams have been the dominant method for representing sentences. N-grams are tuples of words, with n denoting the number of words in the tuple. Since statistical methods estimated probabilities of words given preceeding words, n-grams were a natural choice. However, they manifest the curse of dimensionality. According to [BDV00], given a vocabulary of 100000 words, estimating a joint distribution of 10 words would result in $100000^{10} - 1 = 10^{50} - 1$ parameters to estimate.

Word embeddings were introduced in 2000 [BDV00]. As the author stated, they are „Fighting the Curse of Dimensionality with its own weapons“ by representing words with real valued vectors of size m. The vector represents the word in a more computer-understandable format, where the distance in the m-dimensional space between two vectors represents the grammatical and semantic difference between two words represented by these vectors. The fact leveraged is that similar words will appear in similar sentences, e.g. since the words „dogs“ and „cats“ have a similar probability of appearing in the sentence „Dogs/cats are suitable as pets“. These vectors are generally trained along with the model. There have been different versions of such embeddings proposed, such as word embeddings [BDV00], word-piece embeddings [WSC+16], character embeddings [SMH11] and phrase embeddings [YD15].

Character embeddings have been utilised by [SMH11] by implementing multiplicative RNNs (MRNNs). Most notably, the MRNNs generated text not word by word, but character by character.

The RNN in Eqs. (4) and (5) was modified to take the form of (47) and (48).

$$h_t = \tanh(W_{hx}x_t + W_{hh}^{(x_t)}h_{t-1} + b_h). \tag{47}$$

$$o_t = \tanh(W_{oh}h_t + b_o). \tag{48}$$

For each character embedding, a different weight matrix is learned. The character level embeddings encoded each character as an M-dimensional vector. It achieved state-of-the-art performance when comparing with other character-level models, it falls short when comparing with models which utilise knowledge of entire words.

[YD15] proposed a method for compositing word embeddings into phrase embeddings, called the feature-rich compositional transformation (FCT). For each phrase p, the embedding e_p is constructed using weight vectors λ_i and the individual word embeddings making up the phrase e_{wi} via Eq. (49).

$$e_p = \sum_{i}^{N} \lambda_i \odot e_{wi}. \tag{49}$$

The symbol $\odot$ represents element-wise product, which makes the phrase embedding a weighted product of individual word embeddings. The weight vectors are constructed based on model parameters via Eq. (50).

$$\lambda_{ij} = \sum_{k} \alpha_{jk} f_k(w_i, p) + b_{ij}. \tag{50}$$

$f_k(w_i, p)$ is a feature function that considers the word w_i in the phrase p. The phrase boundaries, tags of speech and heads are based on existing parsers. The feature vector is based on feature templates. The parameters for learning are α. This model is preferred due to its low computational costs.

[WSC+16] introduced word-piece embeddings for use in conjunction with deep LSTMs. This method has become quite popular in recent years due to its enhanced ability to handle rare words. The flexibility of character embeddings [SMH11] is combined with the efficiency of word embeddings. This method completely deterministically generates segmentations for any possible sequence of characters. This is done via a wordpiece model. The task is to select such wordpieces such that when segmenting the corpus according to these wordpieces, the number of segments is minimal. The authors utilised a greedy model to select such wordpieces.

They note that the wordpieces model is the same for the source language and the target language - the model is bilingual. This ensures that the model can unambiguously copy text from source to target, as there are many instances where copying text (such as a phone number of names) is the correct translation. The authors have found that implementation of wordpiece embeddings improved overall BLEU scores.

Word embeddings enabled researchers to utilise neural networks for machine translation. While constant values per each word (e.g. using Huffman coding to assign binary values to tokens) could

work in theory, the problem is that the codes would not encode any information about the words themselves, and the model would have to learn which code means which word. For example, if the word „car" had code 50, word „milk" had the code 600 and the word „shower" had code 550, lots of spurious mathematical relations pop up that can not be utilised, such as $12car = milk$, the distance between „milk" and „car" is „shower".

Word embeddings represent vectors in n-dimensional space. These vectors can be really easy for neural networks to work with, since their inputs tend to be of fixed size, and the framework for performing addition and multiplication is present. Since word embeddings are not fixed, they may be learned alongside the model which utilises them. Therefore, the model will shift the vectors of words deemed statistically relevant to minimise some loss function, encoding the meanings of the words within these vectors during the process.

Word embeddings may be pre-trained, invoking transfer learning. A very popular sort of word embeddings created by [PSM14] was GloVe, which stands for Global Vectors. They leverage overall corpus statistics, such as co-occurrence rates, to generate these representations. The embeddings stem from a noticing that co-occurrences of words relate to their similarity. From a corpus they extracted a probability function for pairs of words, $P(i|j)$, which returns the probability of co-occurrence of word i with word j in the corpus. While the values on their own were not remarkable, the key finding is that ratios of these probabilities are meaningful. The ratio is denoted as follows:

$$r = \frac{p(k|i)}{p(k|j)}, \tag{51}$$

where i and j are words to compare, and k is some arbitrary word. For example, the words „ice" and „steam" are in a sense related (as both of them are forms of water), and are in another sense entirely opposed (one is a gas, another one is a solid). We may choose i to be the word „ice", and j to be the word „steam". The value r will tell us something about the word k. We expect the co-occurrence of „gas" to be higher with „steam" that with „ice", hence that ratio should be relatively low, and it should be opposite for the word „solid". Words that are similarly related or unrelated should have similar co-occurrences, and hence the ratio would be close to 1. It is noteworthy that the authors have managed to show the aforementioned trends for the example words discussed above. This was the basis for their reasoning for constructing the vectors.

They devised a minimisation problem to construct word representations.

$$F(w_i, w_j, \hat{w_k}) = \frac{P_{ik}}{P_{jk}}, \tag{52}$$

where w_i, w_j and $\hat{w_k}$ are vector representations of words i, j and auxillary word k respectively, and P_{ik} and P_{jk} denote the co-occurrence probability of i,k and j, k respectively. F is a function to be determined. Since a feature we want to have in this function is that the differences between vectors are meaningful, the equation may be reworked to be as such:

$$F(w_i - w_j, \hat{w_k}) = \frac{P_{ik}}{P_{jk}}. \tag{53}$$

Since the left hand side of the equation is two vectors, while the right hand side is a scalar, we can produce a scalar on the left hand side via dot product.

$$F((w_i - w_j)^T \hat{w}_k) = \frac{P_{ik}}{P_{jk}}. \tag{54}$$

We want F to be invariant if individual word vectors are shifted around, and for Eq. (54) it is false. To preserve these invariances, Eq. (57) must be true.

$$F((w_i - w_j)^T \hat{w}_k) = \frac{F(w_i^T \hat{w}_k)}{F(w_j^T \hat{w}_k)}. \tag{55}$$

These can be further worked down to take the form of:

$$F(w_i^T \hat{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}, \tag{56}$$

where X_{ik} is the co-occurence between words i and k, and X_i is the occurrence of X_i. This equation works out if $F(x) = e^x$, and the equation transforms to:

$$w_i^T \hat{w}_k = \log P_{ik} = \log X_{ik} - \log X_i. \tag{57}$$

The equation becomes invariant when switching w_i and $\hat{w}_k$ if we include the term for $\log x_i$ in the bias term for i, b_i.

Using these equations, they proposed the final minimisation Eq. (58).

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \hat{w}_j + b_i + \hat{b}_j - \log X_{ij})^2. \tag{58}$$

f is included as a weighting term to weight separate words depending on how often they occur, to make these embeddings more resilient to outliers.

GloVe has gained popularity, as it has achieved state-of-the-art experimental results. One of these tasks is the task of word analogies. The word analogies task involves sentences in the form of „A is to B is what C is to D", where A, B and C are given. For this task, the model ought to find the relation between A and B, and apply that relation on C to produce D. Given word representations as vectors, this can be easily represented via vector additions.

$$w_D = w_B - w_A + w_C. \tag{59}$$

Another type of pre-trained word representations that have overtaken GLoVE are ELMo representations by [PNI+18], which is a feature-based method of transfer learning. These methods use task-specific architectures with pre-trained representations as additional features. ELMo stands for embeddings from language models, and they differ from traditional word embeddings as each token is assigned a representation which depends on the entire sentence.

Language models aim to compute the probability of a sequence of tokens by modelling the probability of the final token given previous tokens.

$$p(t_1, \ldots, t_N) = \prod_{k=1}^{N} p(t_k | t_1, \ldots, t_{k-1}). \tag{60}$$

These language models are known as forward language models, and backwards language models predict sentences in the opposite direction. They aim to model the probability of a token given its following tokens.

$$p(t_1, \ldots, t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, \ldots, t_N). \tag{61}$$

Both of these methods may be implemented as LSTMs. A bidirectional language model combines both a forward and a backwards language model. Essentially, they aim to maximise the value of the following equation.

$$\begin{aligned} \sum_{k=1}^{N} (&\log p(t_k | t_1, \ldots, t_{k-1}; \Theta_x, \overleftarrow{\Theta}_{LSTM}; \Theta_s) \\ &+ \log p(t_k | t_{k+1}, \ldots, t_N; \Theta_x, \overrightarrow{\Theta}_{LSTM}; \Theta_s)). \end{aligned} \tag{62}$$

Θ_x represent the learned token representation, and Θ_s is the learned Softmax layer. $\overrightarrow{\Theta}_{LSTM}$ represent the parameters of the forward LSTM, and $\overleftarrow{\Theta}_{LSTM}$ represent the parameters of the backwards one. The ELMo representation for a token t_k is hence R_k.

$$R_k = \{x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, \ldots, L\}. \tag{63}$$

Where L is the total number of layers in the bidirectional language model, x_k^{LM} is the token representation of said model, and $\overrightarrow{h}_{k,j}^{LM}$, $\overleftarrow{h}_{k,j}^{LM}$ are the internal states of that model. These states are concatenated to give the resulting word representation.

Since the whole sentence gets passed to the bidirectional language model, the resulting representations are context dependent, since they take the whole sentence into account. These representations are vectors, and can be passed as word embeddings to other models.

2.7. Data augmentation in NLP

Data augmentation is the practice of increasing the quantity of data without explicitly collecting new instances. It has been utilised for computer vision, as it is very easy to implement for that domain. For example, when building a cat classifier, one can transform a single cat photo into multiple cat photos by including versions of the same photo flipped, rotated and cropped. This works because an image of a cat still displays a cat after such transformations. This is more difficult for NLP, as word order is often important for the meaning of sentences.

According to [FGW+21], the augmented data must produce data points that are not too similar nor too different from the original dataset. Data which is too similar may contribute to overfitting,

while data which is too different may skew the total training data, making it not representative of the actual data.

There are three branches of techniques for data augmentation: rule-based techniques, example interpolation techniques and model-based techniques. Rule-based techniques utilise predetermined transformations. For example, [ŞS18] borrow the ideas of cropping and rotation from computer vision. Rotation is implemented by choosing a root word and shifting movable word fragments around that word. Dependency trees were used to „crop" sentences by removing peripheral words. This method is advantageous to Baltic and Slavic languages, as the noun cases allow the rotation of words without loss of meaning.

Example interpolation techniques produce new data by taking pairs of data-points and mixing them together. Borrowing from computer vision, this data augmentation spliced one image with a section from another, weighting the labels appropriately to the area of images spliced. Similarly, words of one sentence may be spliced into another sentence. Since there are often patterns in common phrases, a model that has learned a speech pattern ought to be able to generalize it with unseen data. It may be possible to systematically produce different versions of same phrases to aid the model in learning of these templates.

The mixup of different sentences has been proposed by [GKR20]. Assume a source sentence $X \in \mathbb{R}^{s \times V}$ and a target sequence $Y \in \mathbb{R}^{t \times V}$, where V is the size of the vocabulary, s is the length of the input sequence and t is the length of the target sequence. A second training sample $(X^{'}, Y^{'})$ is produced by taking another sentence, and padding or truncating it to match the lengths of (X,Y). A binary combination vector $m = [m_X, m_Y]$ is produced by sampling from the Bernoulli distribution with parameter λ, which is sampled from the beta distribution. The combination vectors determine whether to take the ith token from sequence X or $X^{'}$. Formally, the new data point is produced as such:

$$\begin{aligned}(\hat{X}, \hat{Y}) = (m_X \odot X + (1 - m_X) \odot X^{'}, \\ m_Y \odot Y + (1 - m_Y) \odot Y^{'}).\end{aligned} \tag{64}$$

These sentences will likely not correspond to sentences produced by natural language, but they will contain valid subparts which will force the model to process the models compositionally.

Model-based techniques utilise trained models to produce more data. Generally, pre-trained models are utilised to produce new sentences. Corruption and reconstruction functions for data augmentation have been utilised by [NCG20]. The corruption function masks some words of a sentence, and the reconstruction function tries to predict the masked words. While the corruption function is trivial, BERT [DCL+18] was used for the reconstruction function. BERT is incredibly good at this task, as one of the pre-training tasks for it has been the Cloze task, where a model has to predict masked inputs. This method essentially produces paraphrased versions of the same sentence.

Pre-trained word embeddings have been utilised by [NTW+20] for augmenting data for named entity recognition (NER). For each token $x_i \in X$ in a sequence, m words most similar to x_i are

gathered.

$$C_i = c_{i,1}, \ldots, c_{i,m}. \tag{65}$$

The similarity is determined using pre-trained word embeddings, where cosine similarity is used between the embeddings. These embeddings $c_{i,j}$ are passed through another embedding matrix that is learned to produce $e_{i,j}$. Afterwards these embeddings are processed by the attention unit, which weights the synonyms by their apparent contribution to the sentence.

$$p_{i,j} = \frac{\exp h_i \cdot e_{i,j}}{\sum_{j=i}^{m} \exp h_i \cdot e_{i,j}}. \tag{66}$$

h_i is the hidden state vector for word x_i produced by sentence context embeddings from beforehand. The resulting semantic representation v_i is produced by applying the weights:

$$v_i = \sum_{j=1}^{m} p_{i,j} e_{i,j}. \tag{67}$$

A gate is used to determine how much to consider the synonym representation v_i. The gate works like a gate in an LSTM.

$$g = \sigma(W_1 \cdot h_i + W_2 \cdot v_i + b_g). \tag{68}$$

The gate g determines how much of which input is passed on to the model.

$$u_i = (g \odot h_i) \oplus ((1 - g) \odot v_i). \tag{69}$$

2.8. NLP in inflected languages

There is a body of research of performing NLP for inflected languages, such as Lithuanian. [ŠSR+21] have constructed a financial sentiment dataset collected from Lithuanian news websites, and published it on Kaggle [1]. Then they performed sentiment analysis of these financial statements. Sentiment analysis is a form of single-label text classification, where the label space is usually very small. In this instance the label space consisted of positive, neutral and negative labels. These labels correlate to the sentiment expressed in the sentences to which they are assigned. The authors used several common NLP methods, such as the Naïve Bayes classifier, support vector machine and LSTM, and performed hyperparameter optimization. They achieved accuracies from 59.7% to 71.1%.

A more complicated task of text classification was undertaken by [KK14]. The task is considered more difficult due to the greater number of targets and tentative relation between speech and party ideology.

The task involved classifying Lithuanian parliamentary speeches by their ideology, which are assumed to correspond with their party membership. Given a text document $d \in D$, where D is the

[1] `https://www.kaggle.com/datasets/rokastrimaitis/lithuanian-financial-news-dataset-and-bigrams`

entire document space, return the class $c_i \in C = \{c_1, c_2, \ldots, c_N\}$, which represents the political party of the author of the document. This problem boils down to a single-label classification task, as each document can only belong to one class, since parliament members can only be in one party at a time. The authors aim to produce a classifier $\eta : D \to C$. A Naïve Bayes Multinomial model has been used for classification. This resulted in estimating probabilities outlined in Eq. (70), where n_d is the number of tokens in the document d, and t_k is the kth token in the document.

$$P(c|d) \propto P(c) \prod_{k=1}^{n_d} P(t_k|c). \tag{70}$$

The probability $P(c)$ represents the probability of the class being c, and is computed as $\frac{N_c}{N}$, where N_c is the number of documents belonging to class c, and N is the total number of documents. $P(t_k|c)$ is calculated as $P(t_k|c) = \frac{count(t_k|c)+1}{count(c)+|V|}$, where $count(t_k|c)$ is the number of tokens t_k belonging to documents in class c, $count(c)$ is the number of tokens in class c and $|V|$ is the size of the vocabulary.

They achieved accuracies of 0.54 and 0.49 on two datasets, outperforming the random and majority baselines by 0.279 and 0.13 respectively. They have found that bag of words and bigram interpolation act as the best feature types for this task.

Due to the usage of the naïve Bayes multinomial model, it may not be augmented using the proposed grammatical case submodel. Therefore, a separate model may need to be created for this task. An interesting result would be to compare neural methods with the already explored methods to see the resulting performance difference.

A similar statistical-method based study was performed by [MKM15], aiming to predict the topic of parliamentary debates using their titles and text. For data representation they use the bag-of-words (BOW), n-grams and term frequency-inverse document frequency (tf-idf), and the classifiers were Support Vector Machines (SVM) and multinomial logistic regression. SVMs and multinomial logistic regression seemed to fair similarly.

An SVM estimates a hyperplane in the form of Eq. (71) to split a vector space in two, separating data points belonging to separate classes.

$$y = f_\theta(x) = b_0 + \sum_{j=1}^{k} b_j w_j. \tag{71}$$

$\theta = (b_0, b_1, \ldots, b_k)$ represents the parameters to be learned, while $x = (x_1, x_2, \ldots, x_k)$ represents a document as a vector, where x_i is the count of the ith n-gram in the document. It is evident that the word ordering is lost using this method, essentially only considering the frequencies of words. It is also evident that a single hyperplane may only split two classes, therefore multiple hyperplanes may be needed to learn multiple classes.

Multinomial logistic regression expands upon linear least squares fit (LLSF) method. LLSF is reminiscent of a single neuron, computing predicted classes as in Eq. (72), where A and b are learned parameters, and X is the vector representation of the document.

$$y(X) = A \cdot X + b. \tag{72}$$

Multinomial logistic regression tries maximising likelihoods, computed in Eq. (73). As $A \cdot X + b$ tends to infinity, the closer to $P(y_i|X)$ tends to 1.

$$P(y_i|X) = \frac{e^{A \cdot X + b}}{e^{A \cdot X + b} + 1}. \tag{73}$$

Comparing document representation methods, they have found that tf-idf was superior, achieving high F-scores and outperforming BOW and 3-grams. Using variations of the SVM model, they have managed to achieve F-scores of around 0.81. Logistic regression achieved an F-score of 0.793. Afterwards, the document representation was fixed as tf-idf, and variations of the classifier models were examined, with precision, recall and F-score being computed for evaluation. Since SVMs have a lot of parameters to explore (such as different kernels, kernel degree, cost and gamma parameters), four different versions of them were explored. They have not distinguished any particular model to be significantly superior to others.

As with previously discussed work, there is little in terms of direct reuse to be performed. However, as before, it may be interesting to compare their results with ones achieved by neural network-based models.

Neural network-based models have been created, such as one by [UR22], where a tri-lingual model, called LitLat BERT, was trained. It handles Lithuanian, Latvian, and English text. They trained the RoBERTa base model on masked token prediction, and evaluated it on four tasks: named entity recognition, dependency parsing, part of speech tagging and word analogy.

Due to the lower native-speaking population of such languages, the choice for datasets is restricted when comparing to English. An example for such a dataset is MATAS 1.0 [RBD+19], which is a manually checked morphologically annotated corpus. Despite not having a paper associated with it, it has extensive documentation describing the dataset. A morphological tagset called ”Jablonskis” has been created and is used to describe the words in the dataset. Identical parts of speech have the same number of tags, i.e. all nouns have 7 codes. This is information essential for parsing the tags in a machine-readable format.

Languages other than Lithuanian have also been examined. An NLP platform for the Turkish language have been constructed by [Ery14]. They have implemented tokenizers, de-ascii-fiers, vowelizers, spelling correctors, normalizers, classifiers for whether text is Turkish, morphological analyzers, morphological disambiguators, named entity recognizers and dependency parsers. Quite a few elements, such as de-ascii-fiers, vowelizers, spelling correctors and normalizers are aimed at processing „dirty” language, which is often observed online. For example, vowelizers aim to restore vowels from words which are shortened in language online. An instance of that would be transforming the word „svyrm” to „seviyorum”. A Lithuanian equivalent may be transforming the token „nk” as „niekas”, although the grammatical case of the word would need to be inferred from context since the shortened version of the word does not contain that information. Deasciifiers aim to replace characters which in tend to have diacritics, as generally people tend to omit them when writing while using a keyboard. This is a fortunate overlap of Lithuanian and Turkish languages,

since they both use diacritics. Implementing these features may be beyond the scope of the current work; however, one should be mindful of their existence.

They also utilise morphological parsers, which are going to be essential in the proposed submodel, since extracting morphological information is going to be the purpose of it. A version of [LSP09] was implemented.

Morphological parsers for Malayam-Tamil machine translation have been created by [JRR+11]. They use suffix stripping methods, which, given a word, strips suffixes from the word until the root is left. In inflected languages, such as Lithuanian, suffixes tend to encode morphological information about the word, with the root of the word containing the main, general meaning.

An open source implementation of morphological analysis of the Finnish language has been created by [LSP09] using finite state transducers. Finite state transducers are automata that operate on two tapes - an input and an output tape, and map between two sets of symbols.

Morphological analysers for Arabic have been built by [AEY+08] by constructing multiple separate dictionaries for prefixes, roots and suffixes. Given an Arabic word, it looks up possible combinations using the dictionaries, applies rules for Arabic grammar and provides possible combinations of morphemes that resulted in that word. These morphemes may then be used to extract relevant information.

2.9. Literature summary

The literature review has provided comprehensive understanding of the topic. Following the history of the field from statistical methods in the 1990s, underlying topics for recurring themes, such as the rationale for the attention mechanism and the self-attention model became self evident. The effect and reason of various incremental improvements, such as the development of the LSTM from the RNN and the reasoning behind GRUs have become clear.

The analysis of existing NLP tasks has given many potential tasks to examine the performance of a model. Since the proposed submodel is not task specific, many of the tasks analysed, such as question answering, textual entailment, semantic role labelling, Cloze task and Next sentence prediction may be useful. In particular, the task for semantic role labelling may be useful in constructing the grammatical case submodel itself, as it will have to be trained separately from the model.

Examination of data augmentation for NLP has provided several tools to better utilise data. While datasets have been found, they may be not large enough for our needs. Most notably, the idea of cropping sentences using dependency trees is highly advantageous for languages that will be examined later on. However, since producing dependency trees is not trivial, some of these methods demand the use of other models, which adds another point of failure in the pipeline.

Investigating existing research has shown the state of the research in topics adjacent to the currently examined one. It has given confidence that the particular idea of a grammatical case submodel has likely not been researched, as a lot of the existing work have utilised statistical methods that are now outdated. Hence proposed work may be valuable in exploring the viability of such an idea. While these works may have been fruitful in providing existing datasets, the fact that the

methods used are not based on neural networks mean that a faithful comparison with them is not possible to perform.

The following work will aim to examine the efficacy of a grammatical case submodel. A neural network for inferring a case of a noun will be constructed and trained using an annotated corpus by [RBD+19]. Afterwards, two tasks will be attempted. A regular neural network and a neural network with the grammatical case submodel will be trained for the task of speech classification [KK14]. Then an LSTM with and without the grammatical case submodel will be constructed for the task of financial sentiment analysis [ŠSR+21]. Several runs will be performed to see the average results of models with and without the grammatical case submodule. The accuracies at particular epochs will be compared for model evaluation.

3. Methods and results

The hypothesis that including grammatical case with word embeddings will improve model accuracy will be tested on the financial sentiment analysis task and the parliamentary speech classification task. Each task will be solved with a different model due to the difference of the tasks. Each of the models will be modified to include grammatical case, and the tasks will be solved several times with different seeds to determine the average effect of the grammatical cases during training. Grammatical case will be obtained using a neural network, which will be trained for the task of returning the grammatical case of a word.

3.1. Datasets

3.1.1. Morphologically annotated corpus

A large dataset in form of a mapping of words to grammatical attributes is MATAS, compiled by [RBD+19], on which data analysis was performed. It was decided to compute the relative frequencies of each grammatical case using the following equation:

$$f = \frac{N}{\sum_{0 \leq i \leq 9} N_i}. \tag{74}$$

f is the relative frequency vector, N is the vector containing the total number of words in each noun case, and N_i is the ith value of vector i, meaning the number of words in the dataset that belong to the ith noun case. The results are shown in Figure 1.

As expected, the most common words are those that do not have a grammatical case. It may seem odd that only approximately 28% of words do not have a grammatical case. While there are many possible combinations of words that form sentences, the most basic sentence requires a subject, object, and verb. The subject and object are often nouns, although it is possible to omit the subject or the object. Furthermore, sentences can be enhanced using other parts of speech, such as adjectives, pronouns, adverbs, et cetera. Since in the most basic sentence only a third of the words will not have grammatical case, the value of 28% does not seem out of the ordinary.

It seems interesting that less than half of a percent of all words are in the vocative and illative cases. The illative case is considered an archaic case not commonly used in modern language. The vocative case is used to address somebody, yet its low frequency is harder to explain.

It is fortunate that the dataset is not dominated by a single class, as the model will not achieve low loss for predicting a single class. However, the very low frequencies of the vocative and illative cases are a slight cause for concern, since the model may have difficulties learning those particular cases.

3.1.2. Parliamentary speech data

The task of gathering data for the parliamentary text classifier is a separate problem to solve, as neither [MKM15] nor [KK14] provide the dataset files – they only provide URLs [2] to the offi-

[2] `https://www3.lrs.lt/pls/inter/w5_sale.kad_ses`

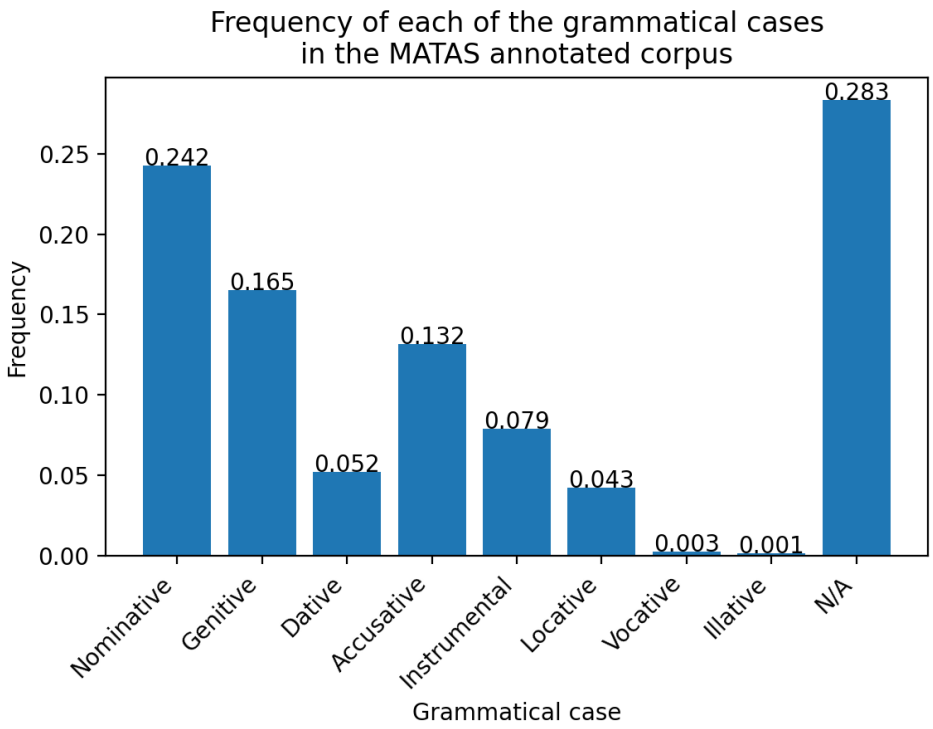


Figure 1. The composition of the grammatical cases in the corpus annotated by [RBD+19].

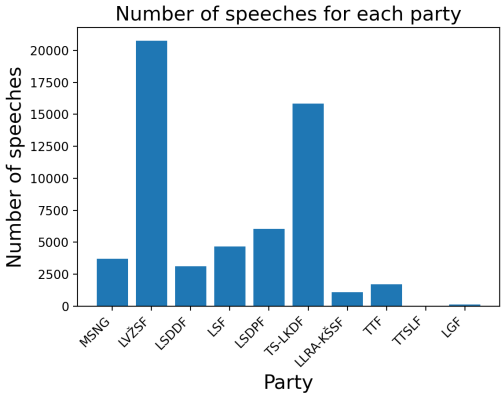
cial Lithuanian Parliament website. Therefore a substantial amount of work has been put into the construction of a scraper to automatically download all transcribed speeches. Furthermore, since those speeches were in the Windows Word binary file format, which are difficult to work with using Python, additional code was needed to convert them.

After the data was gathered, it was decided to use the speeches of the twelfth Seimas of Lithuania, which was in term from 2016 to 2020. It was decided to only use speeches of a single parliament term, since it was expected that the ideology reflected in speeches would most match the parties that produced those speeches.

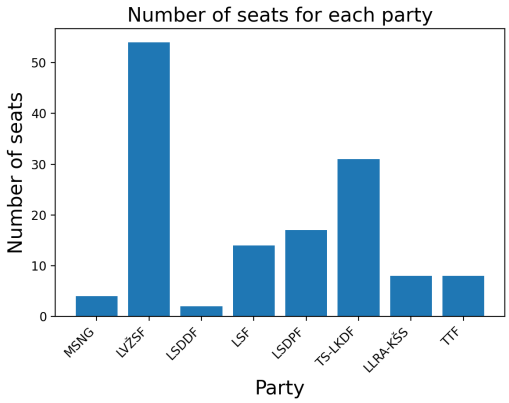
Choosing multiple terms poses several issues. There could be situation that several parties have very similar ideologies due to the fact that the party has changed its name. Furthermore, party members have changed parties, and while this can happen in the middle of a parliamentary term, we assume that it is less likely in a single term than over multiple terms.

Another reason for choosing the speeches of the twelfth Seimas of Lithuania is that it is the latest parliamentary term which has concluded. Choosing the speeches of the thirteenth Seimas of Lithuania was decided against as it has not yet concluded, and therefore the number of speeches is smaller. There is no strong reason for choosing the latest parliamentary term instead of an earlier one besides the expectation that it would be easier to analyse the results for laymen, as the politics of 2016–2020 are the most recent and relevant.

The number of speeches for each party are shown in Figure 2a for a total of 57131 speeches. The party acronyms are: MSNG for mixed group of members of the Seimas (Mišri Seimo narių grupė), LVŽSF for Lithuanian Farmers and Greens Union (Lietuvos valstiečių ir žaliųjų sąjunga), LSDDF



(a) Number of speeches per party.



(b) Number of seats won for each party during the 2016 election.

Figure 2. Speech number statistics.

for Labor faction of the Lithuanian Social Democrats (Lietuvos socialdemokratų darbo frakcija), LSF for Liberals' Movement (Liberalų sąjūdis), LSDPF for Social Democratic Party of Lithuania (Lietuvos socialdemokratų partija), TS-LKDF for Homeland Union – Lithuanian Christian Democrats (Tėvynės sąjunga – Lietuvos krikščionys demokratai), LLRA-KŠSF for Electoral Action of Poles in Lithuania – Christian Families Alliance (Lietuvos lenkų rinkimų akcija – Krikščioniškų šeimų sąjunga), TTF for Order and Justice (Tvarka ir teisingumas), TTSLF for Order and Justice - Sovereign Lithuania (Tvarka ir Teisingumas Suvereni Lietuva) and LGF for (Frakcija "Lietuvos gerovei").

It can be seen that the number of speeches for each party varies greatly, so it may be fruitful to look at the number of seats won for each party during the 2016 election, as shown in Figure 2b. It can be visually seen that the number of speeches somewhat follow the number of members in that party, which is expected.

It can be observed that some parties in the dataset are not present in Figure 2b. This is because the missing parties formed during the term from the members elected during the 2016 election. This may also explain the reason why the missing parties have very few speeches.

Since the label distribution is very uneven, it was decided to balance the data. The balancing would be done by sampling each of the parties for texts up to a particular quantity, which would result in each of the bins being of the same size. The question then is what that quantity should be. Finding the party with the smallest number of speeches and then sampling that number of speeches from all other parties would result in a completely uniform distribution. The drawback of that would be that the dataset would shrink severely. Therefore the quantity chosen was the median of the number of all speeches, which is 3427.

Another option for investigation is selecting the two largest parties and reducing the number of speeches of the larger party to match the smaller party. This approach would present a far simpler version of the problem.

The length of speeches is yet another consideration for model design. If the speeches are relatively short, recursive models may be viable for the classification task. Therefore a histogram of

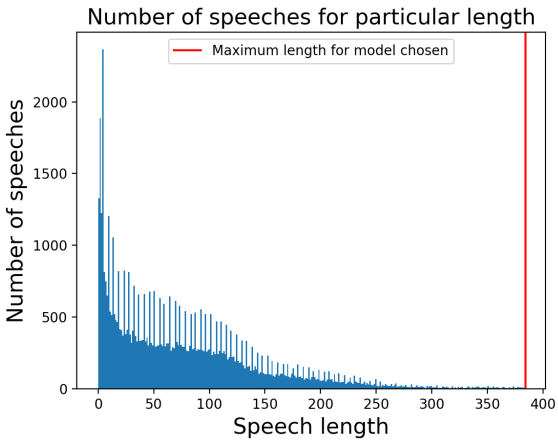


Figure 3. Speech length histogram.

sentence lengths was produced, as shown in Figure 3. Note that the figure is only displayed for the x range between 0 and 400, as there are only 71 speeches that are longer than 400 tokens with the longest one being of 4499 tokens. This has been done to improve the legibility of the figure. Furthermore, the minimum, maximum, mean, and standard deviation of speech lengths have been computed and provided in Table 1. The speech length distribution is highest at the start, and quickly decreases with a rather long tail.

Table 1. Speech length statistics.

Statistic	Value
Minimum	2
Maximum	4499
Mean	89.27
Standard deviation	114.66

3.1.3. Financial sentiment statements

Financial sentiment analysis data has been collected by [ŠSR+21] and published on Kaggle [3]. The dataset consists of sentences collected from articles in business and finance sections of large Lithuanian news websites. Experienced professionals manually classified those sentences into positive (POS), neutral (NEU) or negative (NEG) to obtain the target. The distribution of those classes with their speech lengths are shown in Figure 4. The dataset consists of 2598 negative sentences, 1997 neutral and 5780 positive sentences for a total of 10375. The statistics of the speeches can be found in Table 2. This dataset is a noteworthy alternative for parliamentary speeches due to the shorter sentence lengths.

[3] https://www.kaggle.com/datasets/rokastrimaitis/lithuanian-financial-news-dataset-and-bigrams

Table 2. Speech length statistics.

Statistic	Value
Minimum	3
Maximum	215
Mean	37.68
Standard deviation	19.66

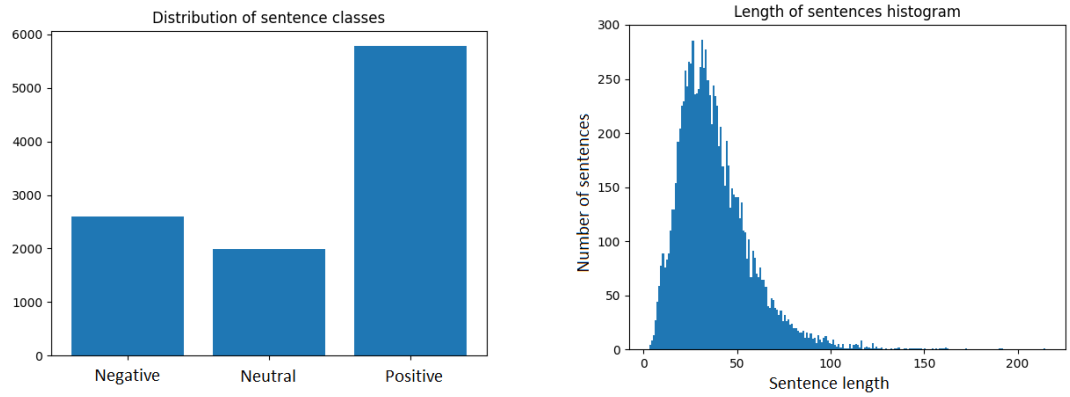


Figure 4. Financial dataset statistics.

3.2. Model construction

3.2.1. Grammatical case model

The computation of noun cases can be done in several methods. They may be taken from a dictionary of word-noun case pairs, or there may be a function that produces a noun case given a word. The dataset compiled by [RBD+19] could be such a dataset – it may be converted to a map, and given a word its case can be looked up in the mapping. This would give quite fast performance, as map lookup in Python is highly optimized. However, a problem arises of what to do if a word is not found in the dataset.

A different approach would be to create a function that finds the noun case given a word. Speakers may quite accurately intuit a noun case of a word just from the letters, as each noun case has a limited set of word endings. For example, all nouns in the locative case end in the letter *e*. A rudimentary algorithm to check for a single noun case would be such: look at the final letter of the word. If the final letter is *e*, then return that the word is in the locative case. However, while the algorithm would return all nouns in the locative case, it is not clear whether or not it would return words that are not in the locative case, but happen to end in the letter *e*. Furthermore, building out such algorithms for each of the noun cases and each of the declension in the lithuanian language would be very labour intensive and error prone.

For this reason it was decided to train a neural network to determine noun cases given a word. This task is easy due to the fact that we have a dataset that has approximately 170000 pairs of word-noun case mappings [RBD+19].

A dataset was formed in the form of tuples containing the word in lowercase and their one-hot

Table 3. The network architecture for the grammatical case model.

Layers	Input Shape	Output Shape
Embedding	(47, 18)	(64, 18)
Flatten	(64, 18)	(1152)
Linear + ReLU (0)	(1152)	(2048)
Linear + ReLU (1-11)	(2048)	(2048)
Linear + ReLU (12)	(2048)	(9)
Softmax	(9)	(9)

vectors. The one-hot vectors were of length 9. The first 8 indices are for the noun cases and 9th is for the case where the word has no case. One-hot vectors were chosen for the fact that it allows the model to guess more generally, as model predictions may be interpreted as probabilities of the word being of a particular case, and then the model can have different confidence levels for its guesses.

The model architecture can be seen in Table 3. The model constructed was a neural network of depth 12 prepended with an embedding layer. The embedding layer maps each letter of the Lithuanian alphabet, as well as numbers and any other characters seen in the dataset, to a vector of length 64. The final layer is a softmax layer, returning a vector of length 9, the values of which sum to 1. This allows the model to predict probabilities for each case since noun cases are mutually exclusive. Since the problem is multi-class classification, the loss used was cross entropy loss. The accuracy for this model is computed as the ratio between the number of words for which the case is predicted correctly, and the total number of words.

3.2.2. Task models

A Naïve Bayes classifier was built by [KK14] and [ŠSR+21] which given a block of text returns the party of its speaker. Implementing a Naïve Bayes classifier may be better for comparison, as both tasks implement them, however, the main purpose of this work is to investigate how a noun case submodule would change the performance of the model. For this reason a neural network is preferable, as it has very few components and it would be easy to add/remove the submodule, which would aid in the comparison between the two versions of the model.

The first step of any model to consider would be to convert the tokens into a form that could be used as input for a neural network. It could be done in either an index, a one-hot vector or a word embedding.

It was decided against using an index, as an index is a single integer. Such an integer implies a quantity, which does not exist in words. It would furthermore imply that algebraic operations are applicable to the words. White it may be true in some sense – the word *lion* and the word *small* may result in the word *cat* – it would require very careful construction of those indices, and such operations would be possible for all – and invalid for most – word pairs.

A one-hot vector would partially solve such an issue – the sum of two one-hot vectors is not itself a one-hot vector, and hence not a word in the dictionary. However, it may seem that we may lose the ability to learn relations between words. That is not true. A linear layer may be represented

as a matrix operation.

$$Y = AX + B, \tag{75}$$

where Y is the result of the first linear layer, A is the weights of the layer, B are the biases and X is the input. If X is a one-hot vector, then the result of the multiplication AX would be the i_{th} row of the matrix A, where i is the position of the element 1 in X. Thus the result Y would be $A_i + B$, where A_i is the particular row of A. This would result in each word being encoded in a vector $A_i + B$, and hence the matrix A would operate as word embeddings, allowing for encoding of relations between words. This would be a good way for encoding the words for the models to use.

Given that the problems call for a sequence-to-one model, a recursive architecture, such as a recursive neural network or long short-term memory network, come to mind. These architectures allow for variable length inputs, which will be very useful since speeches generally vary in length. However, there is a well known drawback: these models have trouble establishing long term relations between words, i.e. if two related words have many words between them, it is difficult for the models to establish relations between them. The difficulty increases with the distance between words, and while LSTMs cope better with such issues, it is still a drawback which cannot be overlooked.

Another currently popular solution would be to use a transformer based architecture, such as BERT [DCL+18]. These models would allow for the utilisation of pre-trained models, increasing performance and decreasing training time. An example of such a model is LitLatBERT [UR22], which is a tri-lingual model for the Lithuanian, Latvian, and English languages. However, there are two potential issues with utilising this model.

The first issue also applies to most transformer-based models – they have a limited input size due to the fact that these models are feed-forward networks. If a speech is longer than a certain length, information may be lost and the model may underperform.

A second issue is more of an engineering issue. BERT models utilise word-piece embeddings. Instead of splitting the text to words and assigning a vector to each, word-piece embeddings split the text by word pieces, and assign each of those a vector. While this is good for model performance – such embeddings may learn word endings that denote noun cases – it would be very difficult to apply noun cases to word-piece tokens, since that would necessitate us to keep track of which word gets split into which vectors, and then applying the noun case to each. Replacing the word-piece embeddings with regular word embeddings is not fruitful, since the word-piece embeddings were trained along with the model, and the pre-trained model would be meaningless without them.

A simpler alternative would be to construct a smaller feed-forward network which uses word embeddings. Such a model would lose the well-established parts of the transformer, such as positional encodings and the attention mechanism, and have diminished performance. On the other hand, it would isolate the noun case component, as other parts would not be able to compensate for the noun cases. Another important benefit of feed-forward models is that it would be faster to train than recursive models, as feed-forward models are far easier to parallelise than recursive models.

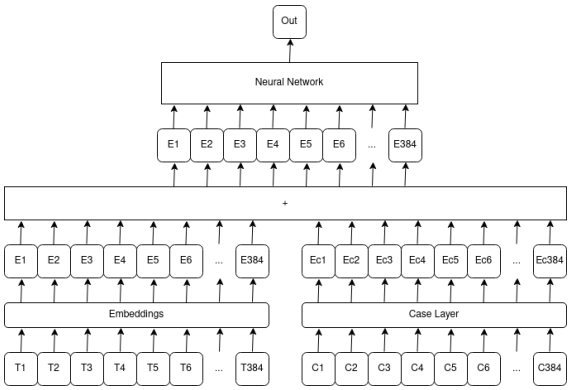


Figure 5. High level description of network architecture.

3.2.3. Parliamentary speech classification model

The important parts of the architecture are described in Figure 5. The first layer is word embeddings, which converts the word tokens into vectors of size 256. Then a separate adjacent linear layer transforms the one-hot case vectors, generated by the grammatical case submodel, into vectors of size 256, so that they could be added to the word embedding vectors. Then these are used by a feed-forward network with 12 layers, where the final layer returns a vector of size 10 – one for each party seen in the term. Those figures are put through a softmax layer, giving probabilities of the speech belonging to each party. Since the problem is also a multi-class classification problem analogous to the grammatical case prediction problem, cross entropy loss is used. Accuracy is computed identically as in the grammatical case model, where the accuracy is the ratio between the number of speeches for which the party is predicted correctly, and the total number of speeches.

The case layer is described as a single linear layer with learned weights $W_c \in \mathbb{R}^{256\times 9}$ and bias $B_c \in \mathbb{R}^{256}$. Given cases in the form of one hot vectors $X_c \in \mathbb{R}^{9\times 1}$, the case vector embeddings $E_c \in \mathbb{R}^{256}$ are computed as such:

$$E_c = W_c X_c + B_c. \tag{76}$$

Then the final embeddings $E \in \mathbb{R}^{256}$ which are given to the feedforward layer are computed via element-wise addition.

$$E = E_c + E_w. \tag{77}$$

A separate linear layer was chosen to expand the case vectors to vectors of size 256 so that it would be trivially easy to remove the grammatical cases without modifying the architecture too much. Furthermore, separating the grammatical case linear layers allows us to inspect the magnitudes of their weights and biases as the model trains.

Not only does such separation allow for enabling and disabling the case layer, but it is quite universal, allowing us to swap out the neural network with another architecture, such as the LSTM for the financial sentiment analysis task.

The feed-forward network was chosen due to speed concerns. Feedforward networks are far

Table 4. The network architecture for the parliamentary speech classification model.

Layers	Input Shape	Output Shape
Embedding	(90001, 384)	(256, 384)
Case Applier (Linear)	(9, 384)	(256, 384)
Linear + ReLU (0)	(256, 384)	(1024, 384)
Linear + ReLU (1-9)	(1024, 384)	(1024, 384)
Flatten	(1024, 384)	(393216)
Linear + ReLU (10)	(393216)	($n_{parties}$)
Softmax	($n_{parties}$)	($n_{parties}$)

more parallelizable, as the whole sentence can be computed in one pass. This is in stark contrast to recursive models, such as the LSTM, where there are as many passes through the network as there are words in the speech. In batch training the number of passes is effectively the number of words in the longest speech in the batch. This is very important for the parliamentary speech classification task, as on average the sentences are very long.

A feedforward network demands a predetermined maximum sentence length. Figure 3 and Table 1 aid in the decision of the number. The original BERT model has a maximum input length of 512. However, that may not be optimal for our purposes, as very few speeches would utilise the additional input length provided by the model. Furthermore, decreasing input length is prudent for us, as a smaller input length allows us to utilise larger batch sizes, speeding up the training process. It was decided to reduce the input length to 384, which is 75% of the original 512 input length used in BERT. This can fully represent 97% of all speeches present in the dataset, which we deemed sufficient. It has been plotted as a vertical red line in Figure 3. Further details of the architecture are shown in Table 4.

3.2.4. LSTM for financial sentiment analysis

The authors [ŠSR+21] provide several of their own solutions – a Naïve Bayes classifier, support vector machine and LSTM. It was decided to utilise an LSTM due to ease of implementation, as it is very easy to swap out the neural network of the parliamentary speech classifier architecture with an LSTM. An LSTM was preferred over a neural network, as it allows a direct comparison with the authors to be performed. This comparison is permitted by the shorter sentence lengths in the dataset, as the sentence length in the parliamentary speech classification dataset is prohibitively long. The architecture details are seen in Table 5.

The overall architecture is akin to one in Figure 5, with the final block replaced with an LSTM. The hyperparameters were made similar to ones the authors used. The authors performed grid search to determine the best values for embedding dimension, hidden unit size and number of tokens, which were 30, 30, and 50 respectively. We have chosen those values to be 32, 32 and 128 respectively as we found that to improve performance.

Table 5. The network architecture for the financial sentiment analysis model.

Layers	Input Shape	Output Shape
Embedding	(45789, 200)	(32, 200)
Case Applier (Linear)	(9, 200)	(32, 200)
LSTM	(32, 200)	(32, 200)
Flatten	(32, 200)	(6400)
Linear + ReLU	(6400)	(3)
Softmax	(3)	(3)

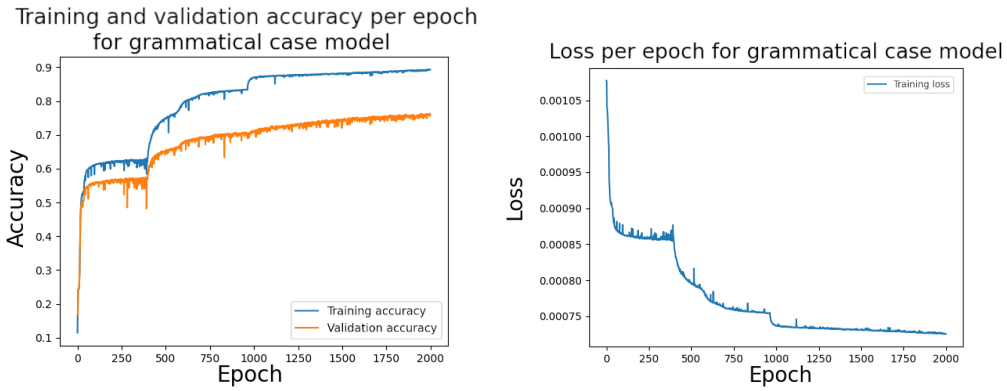


Figure 6. The training and validation accuracy and loss during training of the case model.

3.3. Training

3.3.1. Grammatical case model training

The grammatical case model was trained for 2000 epochs on an NVidia RTX 3060, which took 8 hours of training, and reached an accuracy of 78%, as seen in Figure 6. Note that for the loss graphs only the train loss is visualised, since validation loss was deemed uninformative. It is visible that there were several breakthroughs during the training of the model, as indicated by an increase in accuracy. For the training of the grammatical case model and the parliamentary speech classifier, the training data was split in to training and validation sets with the size ratio of 80% to 20%. The hyperparameters for this training process are listed in Table 6.

3.3.2. Parliamentary speech classifier training

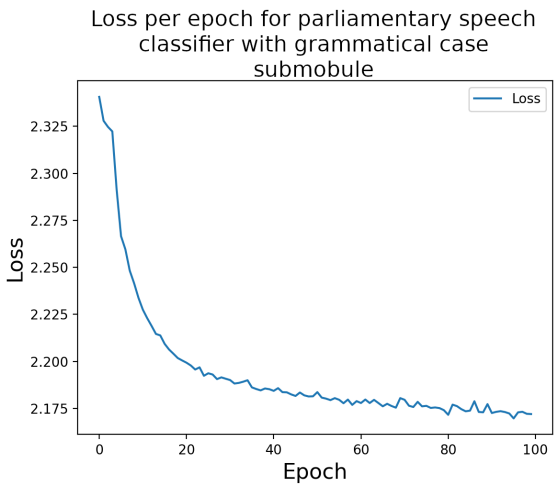
Two versions of the parliamentary speech classifier have been trained – one with and one without the grammatical case submodule. The grammatical case submodule will be deactivated by setting E in

Table 6. Hyperparameters of the grammatical case model.

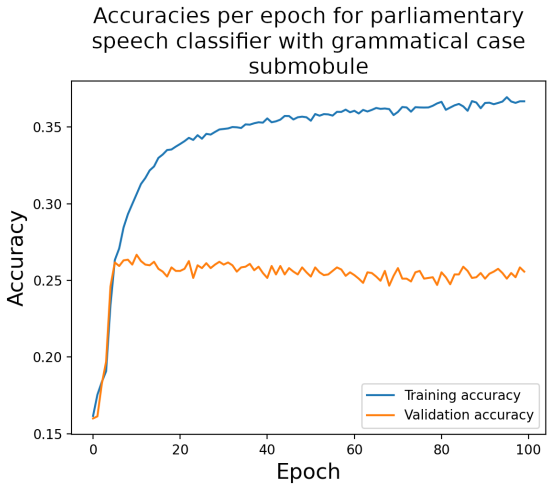
Hyperparameters	Options
Learning rate	0.00001
Epochs	2000
Embedding dimension	64
Number of layers	12
Batch size	2048

Table 7. Parliamentary speech classifier model training hyperparameters.

Hyperparameters	Options
Learning rate	0.00001
Epochs	100
Embedding dimension	256
Number of layers	12
Batch size	32



(a) Loss of full model per epoch.



(b) Accuracy of full model per epoch.

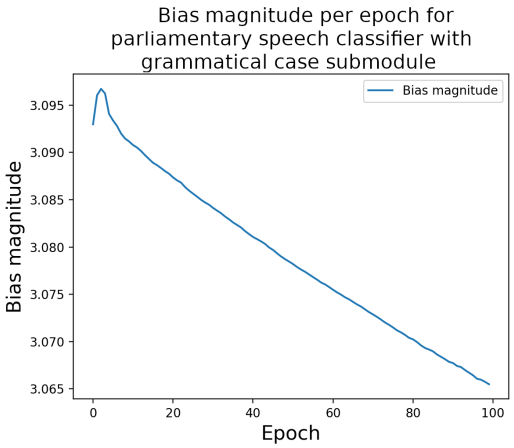
Figure 7. Training statistics for full model.

Eq. 77 to the value E_w. The speech classifier with the grammatical case submodule was trained for 100 epochs which took 8 hours on an NVidia RTX 3060. The loss and accuracy are shown in Figure 7. Other hyperparameters can be found in Table 7. It can be observed that validation accuracy very quickly reaches a hard upper bound, while the training accuracy asymptotically increases. This is a sign of overfitting, as the model fails to generalise to unseen data. The model may associate particular words with political parties rather than the underlying meanings of policy. The simplicity of the model may support this hypothesis, as the model is far simpler than the state-of-the-art transformer architecture, which seemingly achieves understanding of underlying meaning.

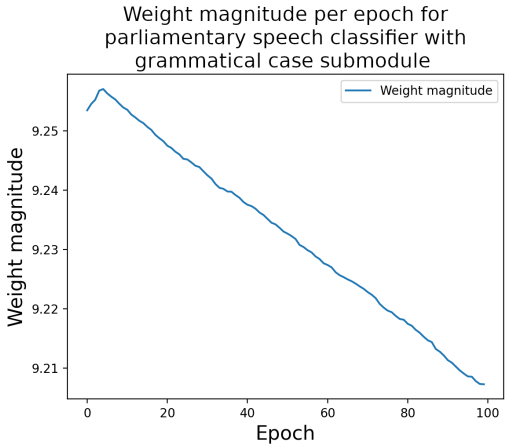
Due to the structure of the neural network, it is possible to visualise the magnitude of the case layer. If the noun cases have an effect on the model, their weights and biases will converge upon a value. This is because there will be a bias to the gradients, which converge the weights and biases towards the optimal value. On the other hand, if the cases are not informative, then the gradients for them will be random, making the weight and bias seem like a random walk. The magnitudes of the case layer are provided in Figure 8.

A copy of the same model was trained, where instead of computing the embeddings E using Eq. (77), the value E_w was set as E. The training was done on the same hardware for the same number of epochs. After the same training process, the training and loss graphs were generated, shown in Figure 9. It was decided to visualize the magnitudes of the case layer in Figure 10. As expected, it is constant since it does not contribute to the computation.

A noteworthy difference of the training process is that the training of the model without the

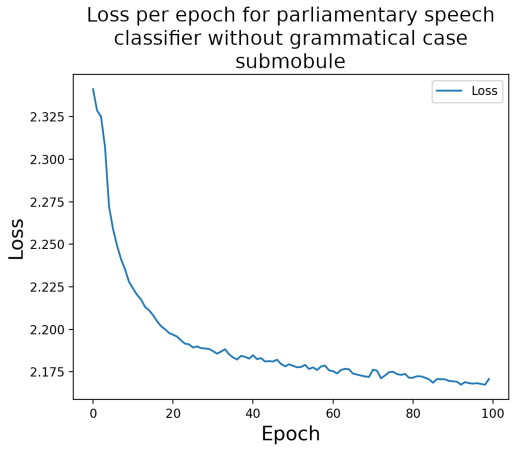


(a) The magnitude of the bias vector of the case layer.

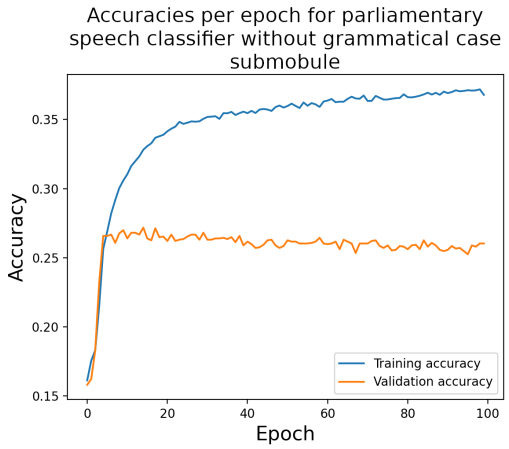


(b) The magnitude of the weight vector of the case layer.

Figure 8. Case layer magnitudes.

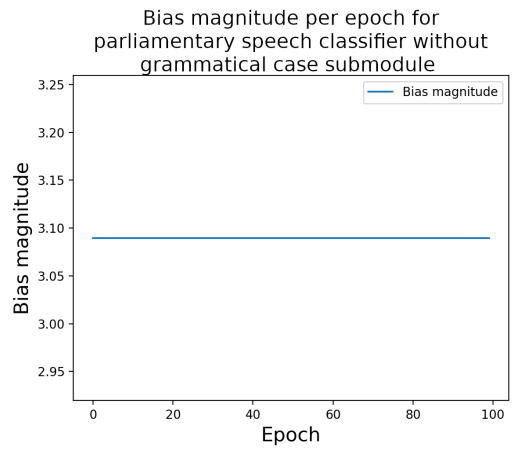


(a) Loss of the model without grammatical case submodule.

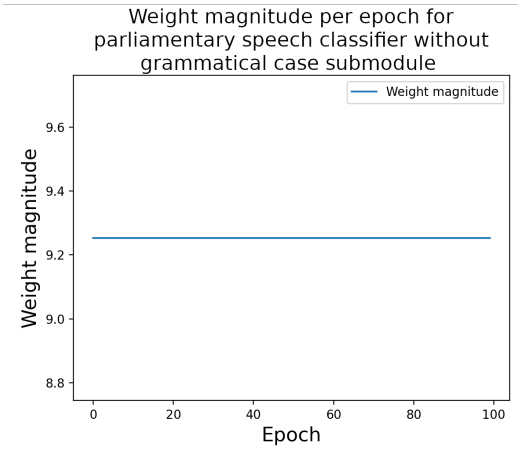


(b) Accuracy of the model without grammatical case submodule.

Figure 9. Training statistics for the model without grammatical case submodule.

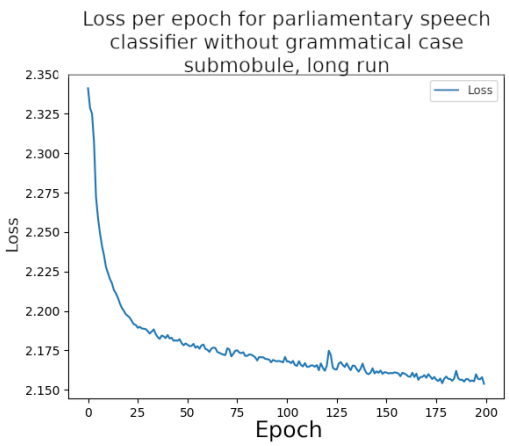


(a) The magnitude of the bias vector of the case layer.

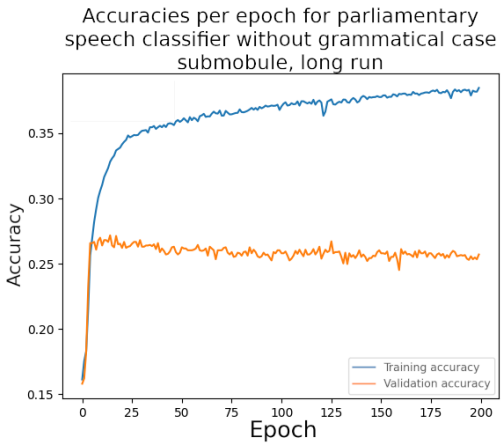


(b) The magnitude of the weight vector of the case layer.

Figure 10. Case layer magnitudes for the model with disabled grammatical case submodule.

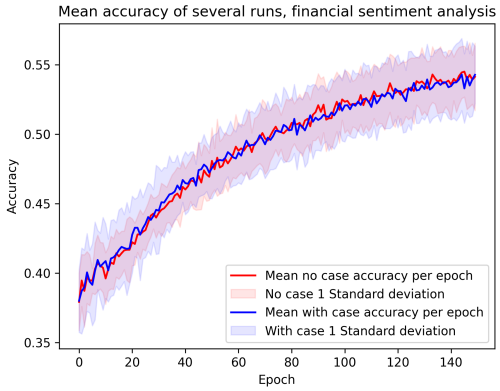


(a) Loss of the model without the grammatical case submodule.

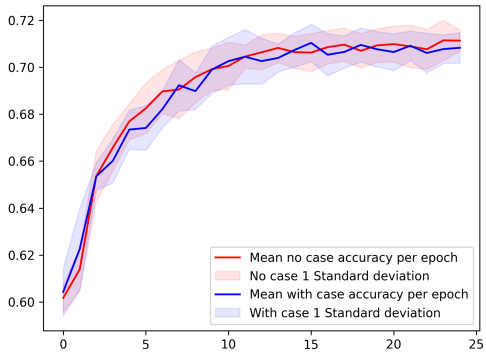


(b) Training and validation accuracy of the model without the grammatical case submodule.

Figure 11. Training statistics for the model without the grammatical case submodule, longer run.



(a) Mean accuracies of several runs for the financial sentiment analysis.



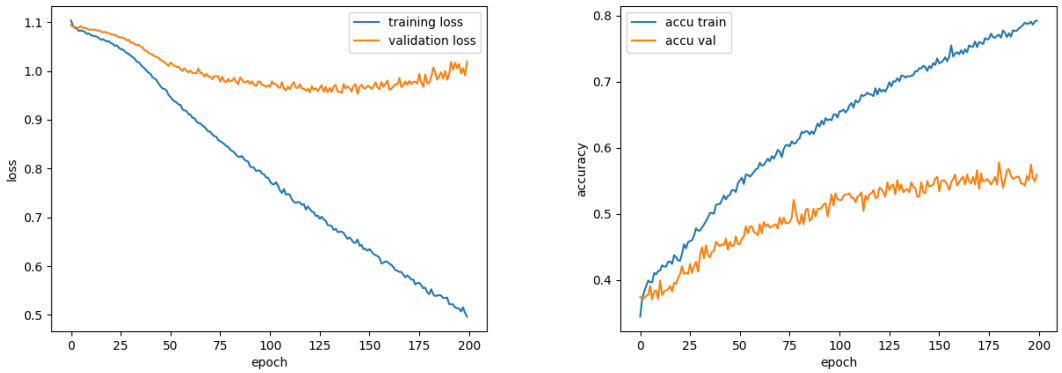
(b) Mean accuracies of several runs for the parliamentary speech classification task.

Figure 12. Comparison of mean accuracies between cased and uncased runs.

grammatical case submodule is twice as fast. For this reason it was decided to train this model for longer to gain insight of whether or not the model would improve with additional epochs. Since the training was around twice as fast, the number of epochs to train the model was doubled in order to match the computation time of the model with that of the grammatical case submodule. The resulting training is described in Figures 11.

It was also attempted to solve the problem by replacing the linear layer with an LSTM. The architecture is similar as described in Figure 5, with the feedforward layer replaced with an LSTM and the number of tokens not set to 384. It was not possible to train this model to any good accuracy, as each epoch took around 2 hours, which is not feasible due to time limitations.

To reduce the effect of randomness, it was decided to repeat the training multiple times. The length of a single run was decided by looking at previous training graphs, such as Figure 11. It seems that on epoch 25 the model is fully trained and begins overfitting, hence each run was limited to 25 epochs to maximise the number of runs, giving greater confidence in the mean values for the validation accuracies. It was chosen to perform 8 runs with and 8 runs without the case submodule,



(a) Loss of the model with the grammatical case submodule.

(b) Training and validation accuracy of the model with the grammatical case submodule.

Figure 13. Training statistics of an exploratory run for financial sentiment analysis.

Table 8. Financial sentiment analysis model training hyperparameters.

Hyperparameters	Options
Learning rate	0.00001
Epochs	150
Embedding dimension	32
Batch size	8

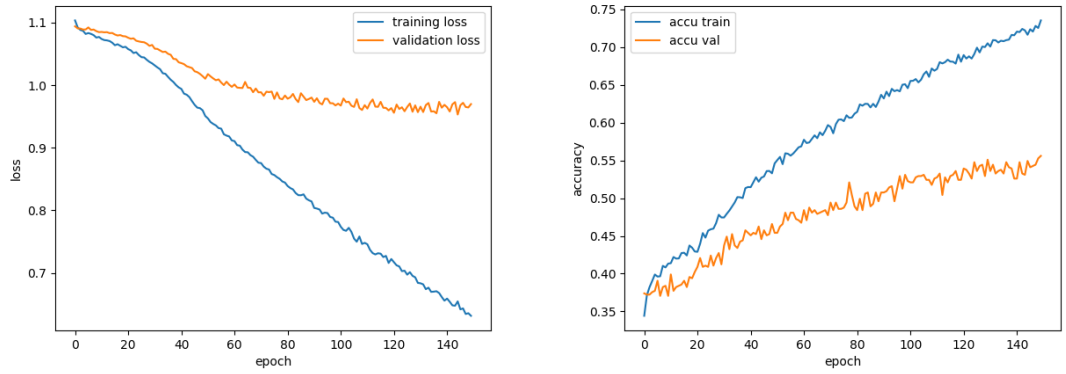
which took 40 hours. The resulting mean validation accuracies of these runs are seen in Figure 12b. It appears that the graphs with and without grammatical case submodule are very similar.

3.3.3. Financial sentiment classifier training

It was decided to evaluate the financial sentiment analysis model using mean validation accuracies of several runs, as that would better show the average effect of the grammatical case submodule. To determine the number of epochs for which to perform those runs, it was decided to train the model for 200 epochs. The graphs for that run can be found in Figure 13. It can be seen that the validation loss reaches a local minimum at around 125 epochs.

The financial sentiment analysis model has been trained 8 times with the submodule and 8 without for a total of 16 runs. Each training was run for 150 epochs, and the final average accuracy achieved was 54.4% and 54.2% for the cased and uncased runs respectively. All 16 runs took around 20 hours on the same hardware as for aforementioned models. The graphs can be seen in Figure 12a. The rest of the hyperparameters are shown in Table 8.

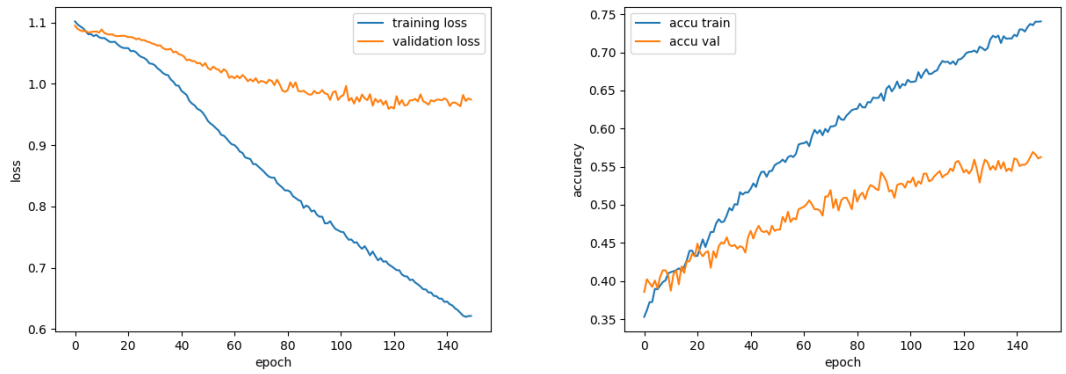
The loss and accuracy of the first run with the grammatical case submodule are shown in Figure 14, while those graphs for the model without the submodule are in Figure 15. Since a minimum in the validation loss is visible, it can be stated that further training would only lead to more overfitting without increases in validation accuracy. As with the parliamentary speech classifier model, the magnitudes of the weights and biases of the case submodule were visualised in Figure 16.



(a) Loss of the model with the grammatical case submodule.

(b) Training and validation accuracy of the model with the grammatical case submodule.

Figure 14. Training statistics of a single run for the model with the grammatical case submodule, financial sentiment analysis.



(a) Loss of the model without the grammatical case submodule.

(b) Training and validation accuracy of the model without the grammatical case submodule.

Figure 15. Training statistics of a single run for the model without the grammatical case submodule, financial sentiment analysis.

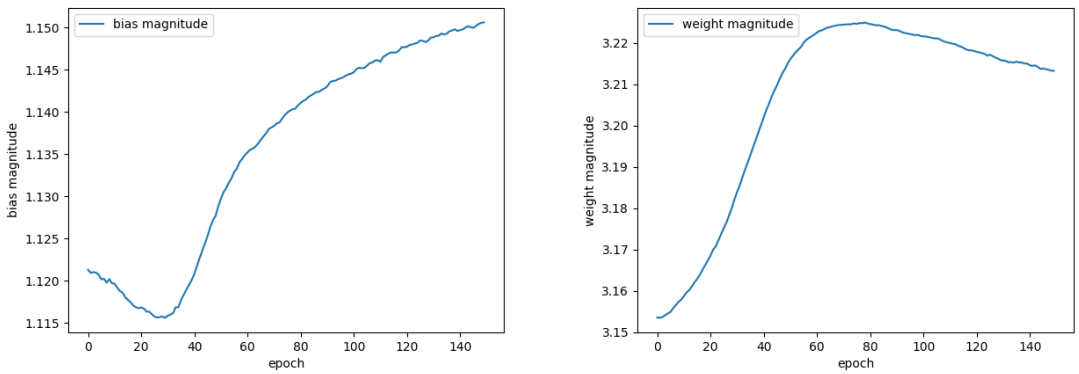


Figure 16. The weight and bias per epoch of the case submodule for the financial sentiment analysis task of a single run.

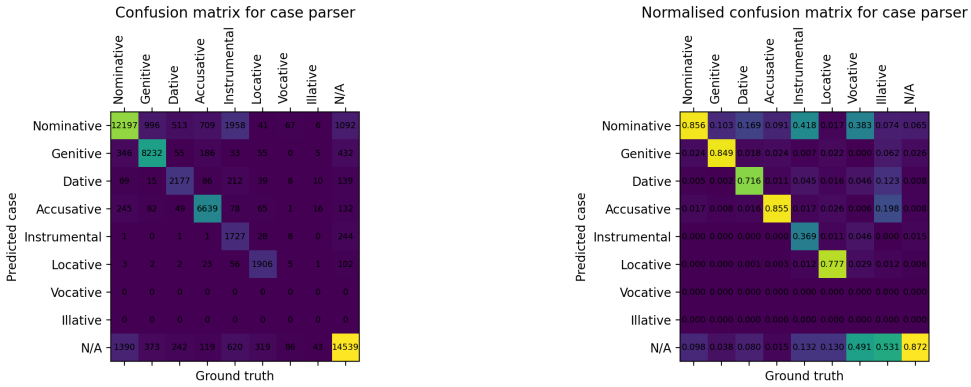


Figure 17. Confusion matrix for trained grammatical case model with the normalised version.

3.4. Model analysis

3.4.1. Grammatical case model results

A more detailed description of the grammatical case model performance can be visualised using a confusion matrix, as shown in Fig. 17. The figure contains the confusion matrix and the normalised confusion matrix. The normalised confusion matrix was generated by dividing each value in the regular confusion matrix by the sum of each column, as described in Eq. (78).

$$A'_{i,j} = \frac{A_{i,j}}{\sum_{k=0}^{9} A_{k,j}}. \tag{78}$$

This essentially gives us the probability of guessing the case at index i given that the ground truth is the case at index j.

$$P(pred = i | GT = j) = A'_{i,j}. \tag{79}$$

This gives us the accuracy for predicting each case, e.g. when the model sees a word in the nominative case, it predicts the nominative case 85% of the times. It allows us to better see how the model behaves.

As expected, the model never tries to guess the vocative and illative cases. Instead it guesses either no case at 49% and 51% of the time respectively. The vocative case can be very easily mistaken with the nominative case with plural nouns, evident with 38% of vocative nouns predicted as nominative, as plural nouns in both cases have identical letters, e.g. nominative plural „Vaikai“ – vocative plural „Vaikai!”.

It is interesting that despite the dative case appearing quite rarely, the model guesses it with an accuracy of 71%. The word endings for this case are one of the following: „-ui“, „-ai“, „-ei“, „-ams“, „-oms“, „-ėms“, „-ims“. This is quite a unique set of word endings, making it easy to distinguish this noun case given the low frequency of these words in the dataset. The only other cases that include those endings are nominative with „-ai“ („apelsinai“), explaining 16% of the predictions for nominative where ground truth is dative.

The instrumental case was quite difficult for the model to learn, reaching only 37%, and guess-

ing the nominative case 41% of the time. Looking at those two cases, they have a few similarities. For example, the word „Citrina“ can be in the nominative or the instrumental case, and the difference in the case is either denoted in the position of the stress in spoken language, or from context in written language. This clash in case only appears in a single declension. Another reason could be that plural nouns in the nominative case differ from their instrumental versions in the final letter, where an ‘s‘ is appended for the instrumental case, such as in nominative plural „Obuoliai“ - instrumental plural „Obuoliais“.

3.4.2. Parliamentary speech classifier results

Comparing the accuracy in Figure 7b with accuracy in Figure 9b, it seems like the grammatical case submodel has little effect on the model. Both validation accuracies seem to approach the value of around 27%, while the training accuracy reaches 36%. While the differences in accuracy between models seem negligible, the best validation accuracy achieved by the parliamentary speech classifier with the grammatical case submodule was 26.7%, while the best accuracy for the version without the submodule was 27.2%. This does suggest that the case layer may be detrimental for the overall model.

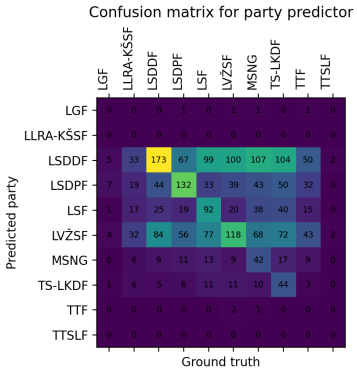
The validation accuracies of the models are not state-of-the-art, but they seem to be better than random. Since six of the parties make up 14% of the total speeches each, a model guessing a single most common party would reach the accuracy of 14%.

Figures 11, which provide losses and accuracies of the longer training of the model without the submodule, show that the training accuracy asymptotically further increases, while the validation accuracy slightly decreases. This is indicative of the model overfitting.

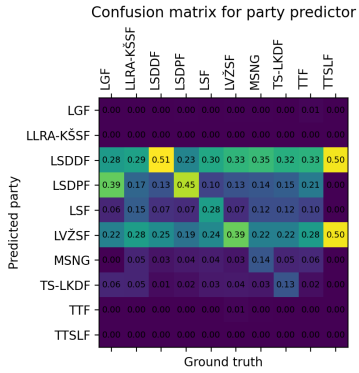
Inspecting Figure 8, it appears that they both start at a random value and then tend towards 0. This may be indicative that the optimal value for weight and bias of the case layer would be zero matrices, removing the influence of the grammatical case submodel. This suggests that the optimal model may be one which does not utilise noun cases.

To gain better insight of how the models learned, confusion matrices were produced. Figure 18 describes the confusion matrices of the parliamentary speech classifier with the grammatical case submodule. As with the grammatical case model, the model entirely avoids predicting parties with fewest speeches. There is a noteworthy difference between the grammatical case model and the parliamentary speech classifier. The grammatical case model avoided predicting two of the cases that had very few words in the dataset. The number of such words makes up around 1% of the entire dataset. On the other hand, while the parliamentary speech classifier avoided predicting parties that were less present in the dataset, the total number of speeches made by non-predicted parties is at around 10%. If the grammatical case model avoided predicting 10% of the least common cases, it would have also avoided the dative and the locative cases. This indicates that a more aggressive balancing with all parties having the same number of speeches may be worth investigating. However, due to the difficulty of the task, the decrease in the total number of speeches to train from would be significant, perhaps diminishing any gained accuracy from the stronger dataset balancing.

For the purposes of comparing the parliamentary speech classifier with and without the gram-

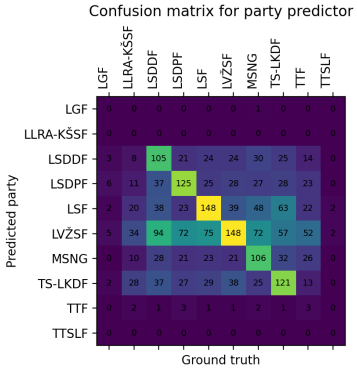


(a) Confusion matrix of the model with grammatical case submodule.

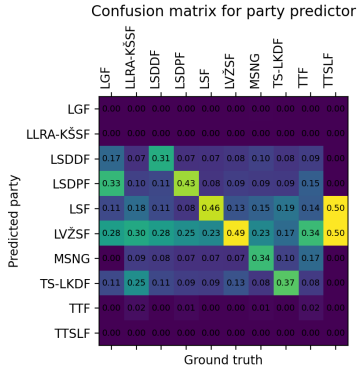


(b) Confusion matrix of the model with the grammatical case submodule, normalised.

Figure 18. Confusion matrices of the model with the grammatical case submodule.



(a) Confusion matrix of the model without the grammatical case submodule.



(b) Confusion matrix of the model without the grammatical case submodule, normalised.

Figure 19. Confusion matrices of the model without the grammatical case submodule.

matical case submodule, see the confusion matrices for the model without the submodule in Figure 19. The confusion matrices for them are very similar, and it does not provide any suggestion that the noun case layer significantly contributes to a different prediction by the model. The normalised confusion matrices in Figures 18b and 19b may seem different on the rightmost column, however, the differences are entirely caused by the unfortunate distribution of speeches in the validation dataset. The validation dataset, as seen in Figure 18 or 19, only received four speeches. Therefore a single different prediction can shift the distribution by 25%.

The lack of effect of the noun case layer on the accuracy or training time may be accounted by the fact that each version of a noun in all different noun cases has its own word embedding. In a well trained model it is reasonable to expect that the noun case may be encoded in the word embedding, making the external noun case obsolete.

The other hypothetical difference could be in training times, as the case module may help the model learn the cases quicker. For this reason Figure 20 was produced, showing the difference between the validation accuracies of models with and without the grammatical case submodule.

The difference in accuracies of the initial run ΔAcc was computed using Eq. (80). A negative

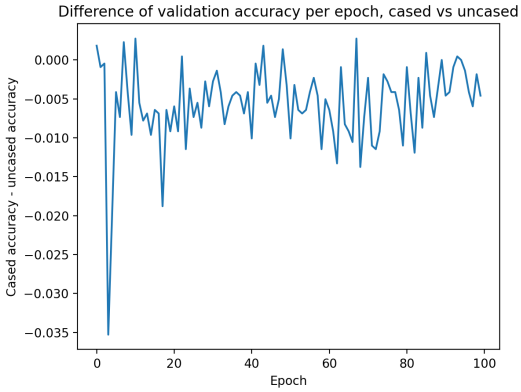


Figure 20. Difference between validation accuracies of cased and uncased model.

ΔAcc notes that the uncased parliamentary speech classifier had higher accuracy, and vice versa.

$$\Delta Acc = Acc_{cased} - Acc_{uncased}. \tag{80}$$

It must be noted that the overall validation accuracy tends to be below zero. The mean of the graph is -0.00568, while the standard deviation is 0.00516. These differences are around 2% of the maximum validation accuracy achieved, which is a very low fraction. However, since the difference was mostly negative, this further supports the idea that the case layer is detrimental for the overall model.

An interesting note is that the difference has a sharp negative peak in the beginning of magnitude 0.0352, after which the difference returns back to around -0.005. This may contradict the hypothesis that a case submodule would require the model fewer epochs to train, as if that were the case, a positive peak in the beginning, showing that the cased parliamentary speech classifier outperforms the uncased model early on, would be expected. Since the peak is in the opposite direction, it suggests that the case module hinders the training early on, and then the model learns to work better with the case submodule. This could explain why the model decreases the magnitude of the case submodule, as seen in Figure 8.

The mean accuracies shown in Figure 12b show very little difference between the model with and without the grammatical case classifier, with the standard deviations being very similar and the mean values being intertwined in the later epochs of training. It is noticable that around epoch 5 and 6 the mean validation accuracy with the grammatical case submodule is almost a standard deviation below the mean validation accuracy without the submodule. This may be used in support of the hypothesis that the grammatical case submodule temporarily hinders the model in the beginning of training, however, the sample size of runs of 8 may be far too small to draw conclusions.

The question arises of why the case submodule may hinder the the training of the overall model. It may be the case that the case submodule is not good enough, and that the 22% of incorrect predictions significantly handicap the overall model.

An accuracy of 54% was reached by [KK14], which is far higher than what has been achieved in this work. There are several differences between their work and ours which make the comparisons less suitable. Firstly, they utilize a lemmatiser, which replaces the different inflections of same

words into the same one. This essentially removes the grammatical cases from the words, as well as makes the data far less sparse. We can not utilise such a lemmatiser, as it would defeat the purpose of the entire work. We attribute the bulk of the difference of accuracies to this step. Secondly, they use the speeches of 2008-2012 parliament session. The party makeup of parliaments, topics discussed and even some beliefs change with time, hence the data used for training may be significantly different. Lastly, they used a Naïve Bayes classifier, which is different from a neural network. While it is generally thought that neural networks outperform statistical methods, the absence of a lemmatiser may outweigh the benefits of a more complex model.

3.4.3. Financial sentiment analysis results

The average accuracy for the financial sentiment analysis model were 54.4% and 54.2% for the cased and uncased runs respectively, which is quite a bit less than the 59.7% accuracy that the authors achieved. This may be due to different hyperparameters that they have not mentioned, e.g. the learning rate.

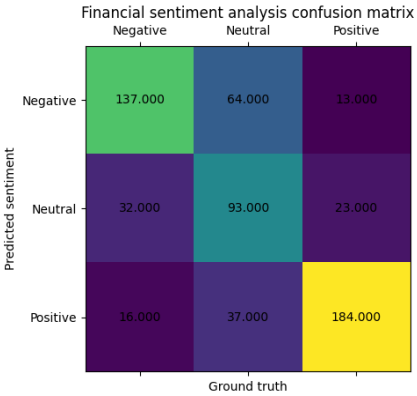
Similarly as in the parliamentary speech classifier, the graphs for the mean accuracy of financial sentiment analysis in Figure 12a seem to be not very informative, although there are more points on the graph to draw conclusions from. The mean accuracy of the models with and without the grammatical case submodule seem intertwined, with differences in the standard deviation attributable to the low sample size. Thus the graph does not show that the grammatical case submodule influences the financial sentiment analysis model in any significant way.

The weights and biases of the grammatical case submodule for financial sentiment analysis, such as in Figure 16, are more informative. The values seem to vary quite a lot, although the absolute values for their variation are less than 0.1. It is not clear whether that difference in amplitude is meaningful. Those values form a curve that looks like an S. It may imply that grammatical cases get learned between epochs 20 and 80, where the change is greatest. However, that epoch range does not seem significant in other graphs of that run.

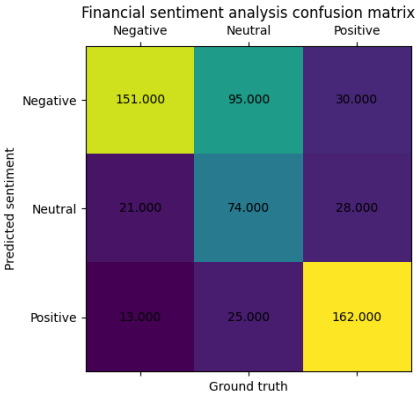
Analogous to the parliamentary speech classifier, confusion matrices were produced. Since there are 8 versions of each model produced, it was chosen to pick the confusion matrices of two best-performing models for comparison purposes. The matrices are shown in Figure 21.

There appears to be little meaningful difference between the confusion matrices. Both of the confusion matrices show issues in misclassifying some neutral sentences as negative. To gain insight into the average performance, all confusion matrices of the same model type were summed to produce cumulative confusion matrices. The confusion matrices are shown in Figure 22.

The confusion matrix in Figure 22a shows that in these experiments, the average model with the grammatical case submodule performs better than the average model without the grammatical case submodule, which can be seen in Figure 22b. This gives slight support for the hypothesis that the grammatical case submodule has merit in a higher overall accuracy.

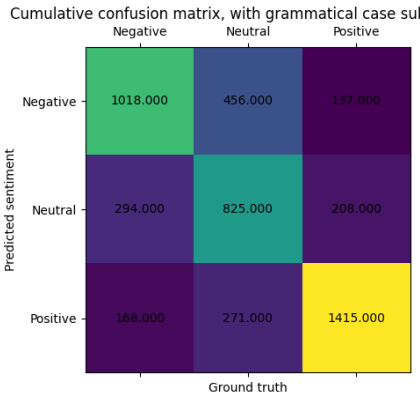


(a) Confusion matrix of the model with the grammatical case submodule.

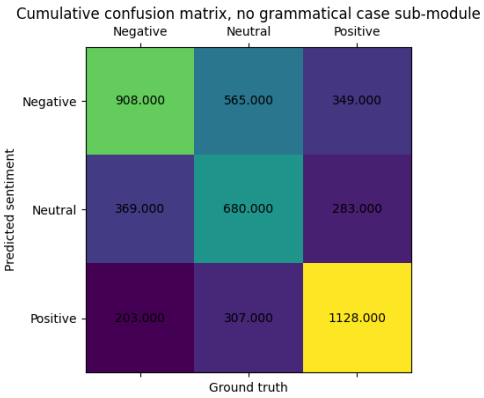


(b) Confusion matrix of the model without the grammatical case submodule.

Figure 21. Confusion matrices of the financial sentiment analysis model.



(a) Cumulative confusion matrix with case submodule.



(b) Cumulative confusion matrix without case submodule.

Figure 22. Sums of all confusion matrices for financial sentiment analysis task.

4. Conclusion

In this work, a model was constructed which can estimate the grammatical case of a noun with an accuracy of around 78%. Then this model was used to augment an LSTM and a feed forward neural network by providing grammatical noun cases of each token in the input. One model was trained on a task presented by [MKM15] and [KK14], which entails classification of parliamentary speeches by the party of the speaker. The augmented model was compared with the version of the model without such augmentations. An LSTM implementation for this task was also attempted, however, no good results were produced due to the very slow training times and the time limitations.

An LSTM was used to solve the task presented by [ŠSR+21], which entails estimating the positive, negative or neutral sentiments of statements regarding finance. Same as with the parliamentary speech classification, the model with and without grammatical case augmentations were compared.

The following are the findings:

- Grammatical case does not seem to significantly aid in the accuracy or training time for the collected parliament speech dataset or the financial sentiment analysis dataset.
- The inclusion of a grammatical model may decrease the performance of the overall model.
- A simple feed-forward neural network may not be sufficient for solving the parliamentary classification task.

The submodule has a fundamental flaw, as it prohibits the use of stemmers or lemmatizers. Thus the datasets are often very sparse, significantly hindering the training process. Word stemming is a popular data pre-processing step which significantly aids in training, thus including the grammatical case submodule in to an existing task will generally lower overall model accuracy, making comparison difficult.

4.1. Future work

It has been noted that due to the lack of lemmatization the datasets are quite sparse, meaning that a single word appears far less often in the dataset due to its particular grammatical case. A possible solution may be constructing pre-trained word embeddings that work with non-stemmed words, explicitly constructed with grammatical case in mind.

This could involve investigating whether word embeddings learned with unstemmed data have particular properties. One of those properties may be the existence of case vectors. These case vectors could be found as the difference between vectors of the same word in different cases; e.g. whether conditions like $x_{berniuko} - x_{berniukas} = x_{obuolio} - x_{obuolys}$, where x_{word} is the vector representation of a word, hold.

There is always a possibility that the methods used were not sufficient – there was no attention mechanism, positional encoding, etc. It may be valuable to analyse how state-of-the-art methods would perform in this task, such as BERT [DCL+18].

It may be valuable to investigate how well the problem of grammatical case classification could be solved similarly to a part-of-speech tagging problem, where the whole sentence may be considered, relieving the need for context for some cases. Furthermore, a feedforward neural network is not a sophisticated solution, and better accuracy may be achieved by considering other model architectures for grammatical case estimation.

Confusion matrices in Figure 18 show some interesting figures, such as the model often falsely predicting the Lithuanian Farmers and Greens Union. While political analysis is beyond the scope of this work, it may be interesting for researchers in political science to investigate the model results.

References

[AEY+08] Lahsen Abouenour, Said El Hassani, Tawfiq Yazidy, Karim Bouzouba, and Abdelfattah Hamdani. Building an arabic morphological analyzer as part of an open arabic nlp platform. In *Workshop on HLT and NLP within the Arabic world: Arabic Language and local languages processing Status Updates and Prospects At the 6th Language Resources and Evaluation Conference (LREC'08)*, 2008.

[ATP+16] Nikolaos Aletras, Dimitrios Tsarapatsanis, Daniel Preoţiuc-Pietro, and Vasileios Lampos. Predicting judicial decisions of the european court of human rights: a natural language processing perspective. *PeerJ Computer Science*, 2:e93, 2016.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BCD+90] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Frederick Jelinek, John Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[BDV00] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

[CFK+20] Ilias Chalkidis, Manos Fergadiotis, Sotiris Kotitsas, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. An empirical study on large-scale multi-label text classification including few and zero-shot labels. *arXiv preprint arXiv:2010.01653*, 2020.

[CFM+19] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, and Ion Androutsopoulos. Large-scale multi-label text classification on eu legislation. *arXiv preprint arXiv:1906.02192*, 2019.

[Che19] Daniel L Chen. Judicial analytics and the great transformation of american law. *Artificial Intelligence and Law*, 27(1):15–42, 2019.

[CVG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[CZL+16] Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.

[DCL+18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DL15] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28, 2015.

[Ery14] Gülşen Eryiğit. Itu turkish nlp web service. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 1–4, 2014.

[FGW+21] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.

[Fre94] Michael Frede. The stoic notion of a grammatical case. *Bulletin of the Institute of Classical Studies*:13–24, 1994.

[GAG+17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pp. 1243–1252. PMLR, 2017.

[GKR20] Demi Guo, Yoon Kim, and Alexander Rush. Sequence-level mixed sample data augmentation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5547–5552, Online. Association for Computational Linguistics, 2020-11. DOI: 10.18653/v1/2020.emnlp-main.447. URL: https://aclanthology.org/2020.emnlp-main.447.

[HK08] Hieu Hoang and Philipp Koehn. Design of the moses decoder for statistical machine translation. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pp. 58–65, 2008.

[Hla00] Barbora Hladká. Czech language tagging (phd thesis) http://ckl. mff. cuni. cz/ hladka/phd_page. html univerzita karlova. *Praha, Czechia*, 2000.

[HLL+17] Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: what works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 473–483, 2017.

[Hoc91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 1997-12. DOI: 10.1162/neco.1997.9.8.1735.

[JRR+11] Jisha P Jayan, RR Rajeev, S Rajendran, et al. Morphological analyser and morphological generator for malayalam-tamil machine translation. *International Journal of Computer Applications*, 13(8):0975–8887, 2011.

[KB13] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1700–1709, 2013.

[KBB17] Daniel Martin Katz, Michael J Bommarito, and Josh Blackman. A general approach for predicting the behavior of the supreme court of the united states. *PloS one*, 12(4):e0174698, 2017.

[KG17] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for lstm networks. *arXiv preprint arXiv:1703.10722*, 2017.

[KK14] Jurgita Kapočiūtė-Dzikienė and Algis Krupavičius. Predicting party group from the lithuanian parliamentary speeches. *Information Technology and Control*, 43(3):321–332, 2014.

[LAS+22] André Lage-Freitas, Héctor Allende-Cid, Orivaldo Santana, and Lívia Oliveira-Lage. Predicting brazilian court decisions. *PeerJ Computer Science*, 8:e904, 2022.

[LFS+17] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, et al. Roberta: a robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[LSD+17] Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. Stochastic answer networks for machine reading comprehension. *arXiv preprint arXiv:1712.03556*, 2017.

[LSP09] Krister Lindén, Miikka Silfverberg, and Tommi Pirinen. Hfst tools for morphology–an efficient open-source package for construction of morphological analyzers. In *State of the Art in Computational Morphology: Workshop on Systems and Frameworks for Computational Morphology, SFCM 2009, Zurich, Switzerland, September 4, 2009. Proceedings*, pp. 28–47. Springer, 2009.

[MKB+10] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, vol. 2 of number 3, pp. 1045–1048. Makuhari, 2010.

[MKM15] Vytautas Mickevičius, Tomas Krilavičius, and Vaidas Morkevičius. Classification of short legal lithuanian texts. In *The 5th Workshop on Balto-Slavic Natural Language Processing*, pp. 106–111, 2015.

[MVW20] Masha Medvedeva, Michel Vols, and Martijn Wieling. Using machine learning to predict decisions of the european court of human rights. *Artificial Intelligence and Law*, 28(2):237–266, 2020.

[MWD+18] James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. *arXiv preprint arXiv:1802.05695*, 2018.

[NCG20] Nathan Ng, Kyunghyun Cho, and Marzyeh Ghassemi. SSMBA: self-supervised manifold based data augmentation for improving out-of-domain robustness. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1268–1283, Online. Association for Computational Linguistics, 2020-11. DOI: 10.18653/v1/2020.emnlp-main.97. URL: https://aclanthology.org/2020.emnlp-main.97.

[NTW+20] Yuyang Nie, Yuanhe Tian, Xiang Wan, Yan Song, and Bo Dai. Named entity recognition for social media texts with semantic augmentation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1383–1391, Online. Association for Computational Linguistics, 2020-11. DOI: 10.18653/v1/2020.emnlp-main.107. URL: https://aclanthology.org/2020.emnlp-main.107.

[PD05] Eric A Posner and Miguel FP De Figueiredo. Is the international court of justice biased? *The Journal of Legal Studies*, 34(2):599–630, 2005.

[PNI+18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics, 2018-06. DOI: 10.18653/v1/N18-1202. URL: https://aclanthology.org/N18-1202.

[PRW+02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[RBD+19] Erika Rimkutė, Agnė Bielinskienė, Virginijus Dadurkevičius, Jolanta Kovalevskaitė, Andrius Utka, and Loïc Boizou. Lithuanian morphologically annotated corpus - MATAS v1.0, 2019. URL: http://hdl.handle.net/20.500.11821/33. CLARIN-LT digital library in the Republic of Lithuania.

[RDU07] Erika Rimkutė, Vidas Daudaravicius, and Andrius Utka. Morphological annotation of the lithuanian corpus. In *Proceedings of the Workshop on Balto-Slavonic natural language processing*, pp. 94–99, 2007.

[RM99] Lance A Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. *Natural language processing using very large corpora*:157–176, 1999.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[SMH11] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 1017–1024, 2011.

[ŞS18] Gözde Gül Şahin and Mark Steedman. Data augmentation via dependency tree morphing for low-resource languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 5004–5009, Brussels, Belgium. Association for Computational Linguistics, 2018-10-11. DOI: 10.18653/v1/D18-1545. URL: https://aclanthology.org/D18-1545.

[SSB14] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling, 2014.

[ŠSR+21] Rokas Štrimaitis, Pavel Stefanovič, Simona Ramanauskaitė, and Asta Slotkienė. Financial context news sentiment analysis for the lithuanian language. *Applied Sciences*, 11(10):4443, 2021.

[Tay53] Wilson L Taylor. "cloze procedure": a new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

[UR22] Matej Ulčar and Marko Robnik-Šikonja. Training dataset and dictionary sizes matter in bert models: the case of baltic languages. In *Analysis of Images, Social Networks and Texts: 10th International Conference, AIST 2021, Tbilisi, Georgia, December 16–18, 2021, Revised Selected Papers*, pp. 162–172. Springer, 2022.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, et al. Google's neural machine translation system: bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[YD15] Mo Yu and Mark Dredze. Learning composition models for phrase embeddings. *Transactions of the Association for Computational Linguistics*, 3:227–242, 2015.