Received July 23, 2021, accepted August 26, 2021, date of publication September 7, 2021, date of current version September 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3110911

Next-Generation Neural Networks: Capsule Networks With Routing-by-Agreement for Text Classification

NIKOLAI A. K. STEUR AND FRIEDHELM SCHWENKER, (Member, IEEE)
Institute of Neural Information Processing, Ulm University, 89081 Ulm, Germany

Corresponding author: Friedhelm Schwenker (friedhelm.schwenker@uni-ulm.de)

Authors thank Ulm University for continuous support. The work of Friedhelm Schwenker is supported by the German Research Foundation (DFG) under Grant SCHW 623/7-1.

ABSTRACT These days, neural networks constantly prove their high capacity for nearly every application case and are considered as key technology for learning systems. However, neural networks need to continuously evolve for managing new arising challenges like increasing task complexity, explainability of decision making processes, expanded problem domains, providing resilient and robust systems etc. One possible enhancement of traditional neural networks constitutes the innovative Capsule Network (CapsNet) technology, which combines the expressiveness of distributed entity representations with an *intelligent* and interpretable signal propagation, named as routing-by-agreement. Since CapsNets represent a relatively young acquirement, further research is essential for gaining profound knowledge about CapsNet theory and best practices for diverse application areas. This paper wants to contribute to the progress of CapsNets for the task of text classification. For this purpose, various research questions about this technology get formulated and experimentally answered with the aid of six selected datasets. In addition, this paper serves as a possible starting point for researchers as well as for practitioners to deal with CapsNets in the text domain, by supplying a survey about its theory, text classification basics and the combination of both areas. The analysis results empirically prove the robustness of CapsNets with routing-by-agreement for a wide spectrum of net architectures, datasets and text classification tasks. Hence, CapsNets can be viewed as a next-generation neural network technology, which offers high potential as text classification method and should be topic of future research.

INDEX TERMS Capsule models, capsule networks, language modeling, neural networks, representation learning, routing-by-agreement, routing procedures, text classification.

I. INTRODUCTION

Since technological advances in computer systems enabled the computation of heavy calculations in relatively small time through massively parallel systems, like mulit-processor computers or Graphical Processing Units (GPU)s, the way was prepared for the neural network technology in many application areas. Common applications for neural networks represent among others computer vision, time series forecasting, anomaly detection, speech recognition and machine translation. Especially, the development of gathering tremendous amount of data through the progressive digitalization in almost every sphere of life, opened nearly unlimited application fields and enabled the training for creating high-performance neural networks. In recent years, neural networks write another success story with the aid of easy to use software libraries (e.g. *TensorFlow* [1]), which bring the capability of this potent technology to almost each company. Despite the powerfulness of contemporary neural networks, the demanded requirements on this technology regularly escalate caused by statutory regulations, expanding problem domains, increasing task complexity, frame conditions with respect to the used datasets etc. In consequence of this, the neural network technology must be continuously enhanced and redefined for satisfying new arising challenges.

One bearer of hope for sustainably enhancing traditional neural networks embodies the innovative Capsule Network (CapsNet) technology, which combines the expressiveness

The associate editor coordinating the review of this manuscript and approving it for publication was Tony Thomas.

of distributed entity representations with an *intelligent* and interpretable communication between consecutive capsule layers. Although the idea behind capsules already arose one decade ago [2], the breakthrough of CapsNets took about six further years and the introducing of an *intelligent* routing between adjacent capsule layers [3]. Because of this long journey for showing the potential of CapsNets, they can be still characterized as a relatively new acquirement in the field of neural networks. However, the CapsNet technology still needs further research to gain knowledge about best practices for dealing with CapsNets in diverse application areas and, specifically, to improve the theoretical understanding of CapsNets and its dynamics.

This paper wants to contribute to the progress of the CapsNet technology in the application field of text classification. In more detail, the contribution of this paper is threefold: Firstly, a comprehensive overview of CapsNet theory including capsule models and diverse routing procedures is provided. Secondly, basic knowledge about language modeling and the task of text classification is supplied. Thirdly, various research questions about different CapsNet configurations are formulated and experimentally investigated in the context of text classification. In particular, this paper should serve as possible starting point for dealing with CapsNets in the text domain for researchers as well as for practitioners. Nevertheless, this paper also covers advanced topics and ideas which hopefully motivate further research about CapsNets.

The structure of this paper is defined as follows: Section *II Fundamentals* begins with the presentation of CapsNet theory and the covering of basic knowledge about text classification. The capsule theory involves the consideration of two capsule models, the construction of capsule layers and the introduction to distinct routing procedures. The text classification basics comprise general characteristics of language modeling, different kinds of representation learning models and a reflection on trends in the area of text classification. Section *III Used Text Datasets* introduces the selected text classification datasets for the analysis part of this paper. Especially, a textual description and some relevant statistics are given for each dataset. Section *IV Related Work* supplies as literature review about the use of CapsNets for the task of text classification. In addition, the most important related works are more precisely described. Section *V Analyses* focuses on the analysis of distinct CapsNet configurations, by defining an appropriate analysis model, describing the training setup and formulating research questions which get experimentally answered. Section *VI Conclusion* finishes the paper with a brief final statement about the potential of CapsNets with routing-by-agreement as text classification method.

II. FUNDAMENTALS

This section contains information about technologies and methods which are relevant for the understanding of the subsequent work. It starts with an introduction to the theory of CapsNets by covering the used capsule models in this paper, explaining the structure of distinct capsule layers and elucidating the routing-by-agreement process as communication behavior between consecutive capsule layers. In addition, different routing procedures are described and discussed. This section ends with the presentation of text classification basics like language modeling, representation learning models and conventional text classification methods.

A. CAPSULE NETWORKS

Hinton *et al.* [2] initially introduced the concept of forming scalar-output neuron groups, named as *capsules*, by proposing a transforming autoencoder which was able to apply a translation operation on a visual entity within an input image. After this work, there passed about six years without substantial progress in the area of capsule technology. Only the work of Sabour *et al.* [3] illustrated the practical use of capsule-based neural networks by establishing an inter-layer dynamic which follows a strategy, named as *routing-by-agreement*. Since this groundbreaking work CapsNets were adopted for various machine learning tasks including image recognition [3]–[7], text classification [8]–[21], knowledge graph completion [22], medical research [23], and many other fields.

1) CAPSULE MODELS

One key idea of CapsNets, compared to conventional neural networks, constitutes the replacement of scalar-output neurons with capsules. Basic properties of a capsule include the encoding of complex entity characteristics in its distributed representation and the ability for quantifying the existence probability of the observed entity [3], [4]. In general, the concept of capsules with both mentioned properties can be realized in various ways. The two capsule models, which will be used in the further proceeding of this paper, are displayed in Fig. 1.

Fig. 1 illustrates the two capsule models according to [3] and [4], respectively. Capsule model (a) is represented as a group of scalar-output neurons, arranged in the form of a vector $\mathbf{p}$. This vector stores the instantiation parameters of the considered entity. For example, in a visual task such an entity could be a low-level feature like a rectangle or a high-level feature like a house. The vector elements in a capsule specify the instantiation of the observed entity in the current input. Obviously, a visual entity like a house can have diverse instantiation parameters such as the thickness of its lines, skewness or hue, but also the number of windows or the type of construction. This intuition about the stored entity properties in capsules for visual tasks analogously holds in the domain of text classification. In that sense, capsules could progressively embrace semantics of phrases, clauses and sentences depending on the considered task. Imaginable instantiation parameters for textual entities may be intensities of certain sentiments, the form of expression or grammar characteristics. But such latent variables in text classification tasks are rarely explored and much more difficult to capture for humans than entity properties in visual tasks. The probability that an observed entity appears in the current input is

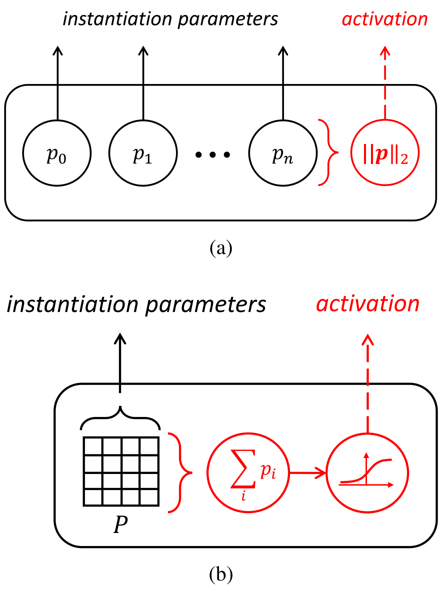

FIGURE 1. Vector-output (a) and matrix-output (b) capsule models, based on the concepts of [3] and [4], respectively. Vector **p** and matrix P are composed of instantiation parameters of the observed entity. The red-colored components illustrate the calculations needed for determining the existence probability of the considered entity.

given by the length of the instantiation parameter vector $||\mathbf{p}||_2$ and is nominated as *activation*. In particular, this definition of capsule activation is biologically plausible since the existence probability of an entity strongly correlates with the intensity of its instantiation parameters, thus, if no entity is recognized with high confidence, the instantiation parameters should also not be determined [3].

Capsule model (b) encodes the instantiation parameters of its observed entity in matrix form. Using a matrix instead of a vector for encoding instantiation parameters, results in practical advantages for recognizing 3-D instances in images, by inherently providing *viewpoint equivariance* through multiplying instantiation parameter matrices with discriminatively learned transformation matrices [2], [4]. The precise dynamic between capsule layers is explained in section *II-A3 Routing-By-Agreement*. Since the effects of matrix-output capsules on tasks with textual data have not been explored yet (to the best of our knowledge), this capsule model was especially selected for further investigation. The activation of capsule model (b) is defined as the sum over all matrix elements, fed into the sigmoid function. The application of the sigmoid function bounds all activations to values in the range of [0, 1]. Therefore, the activation of a matrix-shaped capsule represents a true probability value. To emphasize that activation values are only calculated if they are explicitly requested (e.g. to provide a probability distribution over class capsules in the output layer of a network), the activation output of both capsules are drawn as dotted arrows.

There exist three main reasons for the choice of both capsule models in Fig. 1:

- **Lightweight** and **layer-independent** models
- Learning of **hierarchical** feature **relationships**
- **Parallel attention mechanism** through adjustable inter-layer dynamic feasible.

First, capsule model (a) and (b) can be reduced to models solely containing an instantiation parameter vector or matrix, since the calculation of the capsule activation can be categorized as on-demand functionality. This leads to computationally **lightweight** capsule models compared to other more complex models. For instance, Wang *et al.* [16] proposed a much more comprehensive capsule model for text classification, consisting of internal *representation, probability* and *reconstruction* modules. Moreover, Wang *et al.*'s model [16] was dedicated for the output layer in a network architecture. On the contrary, capsule model (a) and (b) can be easily employed **layer-independent** in a neural network architecture, depending on the realization of the inter-layer communication. Evidently, capsule model (a) and (b) are computationally lightweight related to other more complex capsule models, but consume much more resources compared to an equivalent network architecture based on scalar-output neurons. However, the resulting expressiveness for capturing information about observed entities significantly increases with the use of capsules. Principally, Wang *et al.*'s capsule model [16] can also be used in different layers of a neural network, although the authors indented this model for the output layer, but the main drawback of this model lies again in its required computational effort.

The second argument for the choice of capsule model (a) and (b) thematizes the ability to build **hierarchical relationships** from lower-level features to higher-level ones, which can be characterized as normal behavior within neural networks. Besides, at this point we could just introduce an arbitrary nonlinearity (e.g. an activation function) between consecutive capsule layers and propagate the weighted sum of all capsule outputs from the previous layer to each capsule in the next-higher layer. This would enable the network to learn hierarchical patterns as expected. Actually, one method already exists that utilizes this approach for the inter-layer communication of CapsNets in the field of text classification (cf. *Static Routing* in section *II-A3 Routing-By-Agreement*).

The last reason regards the implementation of the dynamic behavior between capsule layers as **parallel attention mechanism** [3], [8]. For this purpose, Sabour *et al.* [3] firstly introduced the notion of *routing-by-agreement* which fulfills the assignment of part-to-whole relationships. Briefly explained, lower-level capsules encode entities that represent parts to the wholes which are in turn encoded in the entities of the next-higher capsule layer. For instance, in computer vision parts could be a nose, mouth or eyes and the corresponding whole could be a human face. Now, a routing-by-agreement procedure tries to dynamically route recognized parts to their wholes in the next-higher layer. This dynamic should reduce the noise ratio in higher-level capsules and acts like a parallel attention mechanism, because this process is applied for each higher-level capsule [3], [8].

Thus, routing-by-agreement supports the learning of more complex hierarchical relationships. The concrete implementation of a routing-by-agreement procedure can also be strongly problem-specific to exploit expert knowledge in a given domain.

2) CAPSULE LAYERS

One problem that may arise when designing a CapsNet is how to build an introducing capsule layer upon scalar-valued features. A possible solution can be the extraction of instantiation parameters from a preceded convolutional layer. Fig. 2 depicts the conversion of the output from a 2-D Convolutional Layer (Conv2D) into a convolutional capsule layer. A Conv2D was selected for illustration purposes, using a 1-D Convolutional Layer (Conv1D) would follow the same concept. First of all, the feature maps of the convolutional layer are pooled into groups with size corresponding to the number of instantiation parameters d for one capsule. Afterwards, the scalar values at each 2-D position in a group of feature maps are gathered in order to shape an instantiation parameter vector or matrix with d elements. Finally, the resulting capsule feature maps are collected to constitute a convolutional capsule layer. The initial capsule layer within a network is usually referred to as *Primary Capsules (PrimaryCaps)* [3]. To ensure equal-sized feature map groups, the number of feature maps n must fulfill the constraint

$$n \bmod d = 0. \quad (1)$$

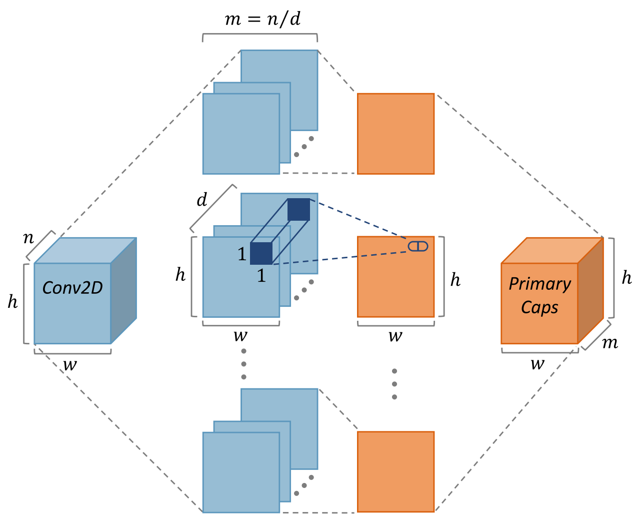

FIGURE 2. Exemplary proceeding for building a convolutional capsule layer upon a 2-D convolutional layer, based on the concept of [3].

Hence, the required number of filters m of the Conv2D is given as $m = n/d$, for providing a desired capsule dimensionality. Spatial information is preserved in the combined capsule-based feature maps. As a consequence, the width w and the height h dimension remain unchanged. The great advantage of concatenating scalar-valued features to instantiation parameter vectors or matrices relies on the exponentially increasing efficiency compared to scalar-valued features [3].

Principally, capsules can be composed of arbitrary scalar-valued features. For example, another possibility represents the conversion of hidden states in a Recurrent Neural Network (RNN) layer into entity instantiation parameters for capsules [13], [16]. Nevertheless, the above presented concept for creating PrimaryCaps based on a regular convolutional layer is one of the most frequent applied variants.

After the composing of PrimaryCaps as initial layer for each CapsNet, conventional neural network layers can be simply transferred into capsule-based versions by rather operating on capsules than on scalar-output neurons. As a remark, capsule layers still follow another inter-layer dynamic than regular neural network layers which must be taken into account for implementation purposes.

3) ROUTING-BY-AGREEMENT

The general routing-by-agreement process can be summarized in five chronological steps: **1.** Each capsule in a lower-level layer predicts the instantiation parameters for all entities encoded in capsules of the next-higher layer. **2.** The instantiation parameters of a higher-level capsule are computed based on the predictions from the preceded layer. **3.** Predictions that are in coherence with the instantiation parameters of a capsule in the next-higher layer are likely to be part of the higher-level entity. **4.** Therefore, the contribution of agreeing predictions to the instantiation parameters of the higher-level capsule gets increased. **5.** This process can be iteratively applied since changing the contribution of predictions, in turn leads to changes in the instantiation parameters of higher-level capsules [3].

Fig. 3 illustrates the sending of predictions from a fully-connected capsule layer to the subsequent one. For instance, the prediction $\hat{U}_{j|i}$ for capsule c_j^l is computed as multiplication of U_i with the corresponding transformation matrix W_{ij}. Using transformation matrices as connection weights, instead of scalar values, results in the ability to

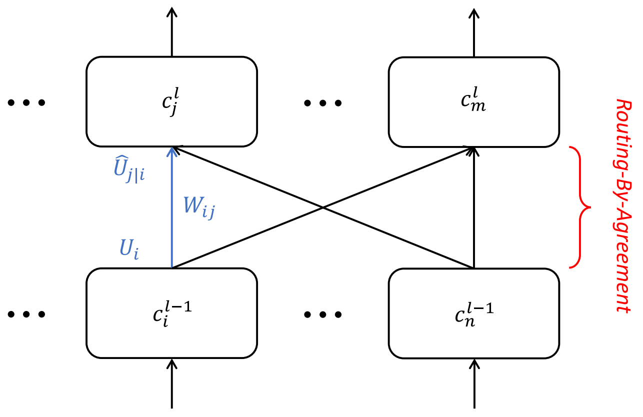

FIGURE 3. Visualization of the general routing-by-agreement process between two fully-connected capsule layers, oriented on formulae from [3].

encode complex spatial relationships between capsules from consecutive layers [3]. The prediction U_i is stated as matrix for supplying a generic mathematical formulation for vector and matrix-shaped capsule models. All predictions of the preceded capsule layer are finally processed by a chosen routing-by-agreement procedure, which specifically weights the collection of inputs for each capsule in the subsequent layer. A prediction can be mathematically expressed as

$$\hat{U}_{j|i} = W_{ij} U_i \oplus b_{ij}. \tag{2}$$

In opposite to Sabour *et al.* [3], the computation of predictions was extended with a connection-specific bias which gets element-wise added to the transformed outputs of preceding capsules. Other approaches frequently use solely matrix multiplication for determining capsule predictions, as Sabour *et al.* [3] initially proposed. Hinton *et al.* [4] also added two learnable biases per capsule but how they were exactly involved was not further explained. Zhao *et al.* [8] assigned an individual bias term to each higher-level capsule that is added to predictions from the preceded layer. Thus, as opposed to Hinton *et al.*'s [4] or Zhao *et al.*'s [8] approach, in this paper each connection between consecutive capsule layers gets assigned an individual bias term.

The bias term in (2) can be interpreted as a dimmer which controls the intensity of predictions. Since the intensity of predictions can directly affect the magnitude of capsule activities in a subsequent layer, depending on the chosen routing-by-agreement procedure, a bias term can amplify the expressiveness of a CapsNet architecture. Another view of such a bias term is to understand it as a memorization mechanism that restricts the hypothesis-space for function approximation based on trained samples [24]. Hence, this modification should improve the convergence behavior of a CapsNet and also speed up its training.

Although a routing-by-agreement process is frequently used as dynamic behavior between capsule layers, the inter-layer communication within a CapsNet can be abstracted to a generalized routing function that owns the ability to fully control the signal propagation from on capsule layer to the next one. Such a routing function takes all predictions $\hat{\mathbf{U}}^{l-1}$ from a preceding layer and determines the instantiation parameters $\mathbf{U}^{l}$ for all capsules in the subsequent layer:

$$\mathbf{U}^{l} = route(\hat{\mathbf{U}}^{l-1})$$
$$with\ \mathbf{U}^{l} \in \mathbb{R}^{m\times c\times d}, \quad \hat{\mathbf{U}}^{l-1} \in \mathbb{R}^{n\times m\times a\times b}. \tag{3}$$

In this paper, capitalized and bold letters in formulae denote the mathematical construct of tensors. A tensor can be described as a numerical array defined on a regular grid with an arbitrary number of dimensions [25]. In this work, tensors are only used for multidimensional arrays with at least three dimensions. The dimensions m and n correspond to the number of capsules in the l and $(l-1)$-layer, respectively. The dimensions a up to d represent the dimensionality of instantiation parameters for capsules in both layers. Again, the capsule dimensionality is in both cases defined as matrix for providing an universal formulation for vector and matrix-shaped capsules. Besides, the distinction of the capsule dimensionality between both layers indicates that capsule dimensionalities can be implicitly modified from layer to layer through the shape specification of transformation matrices:

Case	**Dimensions**	
$Vec \rightarrow Vec$	$(b\times a) * (a\times 1) = (b\times 1)$	(4a)
$Mat \rightarrow Mat$	$(c\times a) * (a\times b) = (c\times b)$	(4b)
$Vec \rightarrow Mat$	$(a\times 1) * (1\times b) = (a\times b)$	(4c)
$Mat \rightarrow Vec$	$(1\times a) * (a\times b) = (1\times b).$	(4d)

In principle, the shape of instantiation parameters can be arbitrarily chosen to be in vector (*Vec*) or matrix (*Mat*) representation per capsule layer. All possible capsule dimensionality conversions between two adjacent layers are illustrated in (4). The required dimensions for transformation matrices are highlighted in red color for each case. To ensure arbitrary conversions, capsule vectors must be sometimes existent as column or row vectors. Furthermore, resulting instantiation parameter matrices may be rearranged in the last instance. However, these additional reshaping operations should not be an important problem from a technical perspective. A relevant property that results from the use of transformation matrices is the capability to shrink or grow capsule dimensionality between adjacent layers on demand, at the time of model definition.

4) ROUTING PROCEDURES

Dynamic routing was simultaneously proposed with the term *routing-by-agreement* [3]. The pseudo code of the dynamic routing procedure is displayed in Listing 1. Dynamic routing is an iterative method that receives as input the prediction $\hat{\mathbf{u}}_{j|i}$ of all capsules in the previous layer and the number of iterations r for applying the routing. Predictions are represented in vector form and their agreement to the current instantiation parameters of a higher-level capsule is measured with the dot product of both vectors. In the end, the procedure returns the new instantiation parameters $\mathbf{v}_j$ for the higher-level capsule j. Hence, this procedure needs to be conducted for each capsule in the higher-level layer.

There are some aspects that should be noticed to dynamic routing. First, the idea of this routing procedure is to repeatedly determine the likelihood that a lower-level capsule is part of an entity represented by a higher-level capsule and, then, correspondingly weight its contribution to this higher-level capsule. A contribution weight c_i is also called *coupling coefficient* [3]. This communication behavior can be seen as a parallel attention mechanism between capsule layers, which tries to filter relevant features [3]. The *squash* function acts similar to an activation function which implements an additional nonlinearity within the network by normalizing capsule outputs between layers. Generally, this increases the learning capacity of the neural network and stabilizes

```
procedure ROUTING($\hat{\mathbf{u}}_{j|i}, r, l$)
    for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$:
        $b_{ij} \leftarrow 0$
    for $r$ iterations do
        for all capsule $i$ in layer $l$:
            $\mathbf{c}_i \leftarrow a_i \cdot leaky{-}softmax(\mathbf{b}_i)$
        for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
        for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow squash(\mathbf{s}_j)$
        for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$:
            $b_{ij} \leftarrow b_{ij} + \langle \hat{\mathbf{u}}_{j|i}, \mathbf{v}_j \rangle$
        return $\mathbf{v}_j$
```

LISTING 1. Pseudo code of the original dynamic routing procedure from [3] with modifications (in red) suggested in [8]; Notation adjusted for listing uniformity in this paper.

```
procedure EM ROUTING($\mathbf{a}, V$)
    $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
    for $t$ iterations do
        $\forall j \in \Omega_{L+1}$: M-STEP($\mathbf{a}, R, V, j$)
        $\forall i \in \Omega_L$: E-STEP($\mu, \sigma, \mathbf{a}, i$)
        return $\mathbf{a}, M$
procedure M-STEP($\mathbf{a}, R, V, j$)
    ▷ for one higher-level capsule, $j$
    $\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * a_i$
    $\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
    $\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij}(V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
    $cost^h \leftarrow (\beta_u + log(\sigma_j^h)) \sum_i R_{ij}$
    $a_j \leftarrow logistic(\lambda(\beta_a - \sum_h cost^h))$
procedure E-STEP($\mu, \sigma, \mathbf{a}, V, i$)
    ▷ for one lower-level capsule, $i$
    $\forall j \in \Omega_{L+1}$: $\mathbf{p}_j \leftarrow \frac{1}{\sqrt{\prod_h 2\pi(\sigma_j^h)^2}} exp(-\sum_h \frac{(V_i^h j - \mu_j^h)^2}{2(\sigma_j^h)^2})$
    $\forall j \in \Omega_{L+1}$: $\mathbf{R}_{ij} \leftarrow \frac{\mathbf{a}_j p_j}{\sum_{k \in \Omega_{L+1}} \mathbf{a}_k p_k}$
```

LISTING 2. Pseudo code of the EM routing procedure [4]; Notation adjusted for listing uniformity in this paper.

signal propagation. According to Sabour *et al.* [3], the squash function can be formulated as

$$squash(\mathbf{s}_j) = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}. \tag{5}$$

This function normalizes vector lengths to be in range of $[0, 1]$, whereas vector orientations are preserved. Obviously, the dynamic routing procedure accomplishes all principles associated with the routing-by-agreement concept, since part-to-whole assignments between capsule layers are supported by checking the consensus between lower and higher-level capsules. This selective choice of input features leads to the advantage of reducing noise ratios in higher-level capsule inputs which increases the overall efficacy of a CapsNet. The main shortcoming lies in the computational effort for repeatedly adjusting part-to-whole assignments that grows with higher iteration numbers. However, Sabour *et al.* [3] experimentally demonstrated that three iterations should be sufficient for fast convergence, while redefining the logits b_{ij} between capsules.

The red-colored parts in Listing 1 highlight two enhancements, proposed by Zhao *et al.* [8]. These enhancements comprise the inclusion of lower-level capsule activity a_i and using a *leaky-softmax* variant, instead of the common softmax function. A possible way for computing activations for vector and matrix-shaped capsules was already explained. Increasing or decreasing coupling coefficients based on lower-level capsule activations is also biologically plausible, because the existence probability of an entity should be entangled with its propagation to higher-level entities (cf. [3]). A leaky-softmax function is applied to allow a slight transfer of irrelevant information through the network, which should help to reduce the routing of noise to relevant higher-level capsules [8]. Regarding text classification, such irrelevant information could be stop words, punctuation marks or just words with less informative content [12], [20]. Moreover, an *orphan category* [3], [8], [12], [20] as an auxiliary output capsule can then capture background noise in the last instance. Since Zhao *et al.* [8] did not explain the concrete implementation of their leaky-softmax function, this paper provides its own realization. This paper uses a leaky-softmax variant which is represented as a two-round softmax function:

$$leaky\text{-}softmax(\mathbf{b}_i) = softmax(softmax(\mathbf{b}_i) * \alpha). \tag{6}$$

The learnable parameter α controls the distance from high-probability values to values that were stated after the first round to be zero. Fig. 4 visualizes an exemplary application of the leaky-softmax function. After the first application of the softmax function in subfigure (b) solely the four logits with the previously dominating magnitudes have probability values higher than zero. But after the weighting with α and the second application of the softmax function, subfigure (c) shows that all zero-probabilities were lifted to values slightly above zero.

Another routing-by-agreement implementation represents **EM routing** [4] which is grounded on the *Expectation-Maximization (EM)* algorithm [26]. The EM algorithm applies alternating an *expectation* and *maximization step* for iteratively computing maximum-likelihood estimates [26], [27]. Especially, the EM method can be utilized as clustering algorithm for Gaussian mixtures, which tries to find Gaussian-like clusters within a collection of data points [4], [28]. The belonging of data points to clusters is realized with *soft assignments* [28].

EM routing adopts the EM clustering algorithm for Gaussian mixtures to assign lower-level features to higher-level ones [4]. The corresponding pseudo code is illustrated in Listing 2. This routing procedure clusters incoming predictions from lower-level capsules to higher-level capsules. Thus, each higher-level capsule defines its own Gaussian-like cluster. Similar predictions from lower-level capsules should be close together in the prediction space and vote for

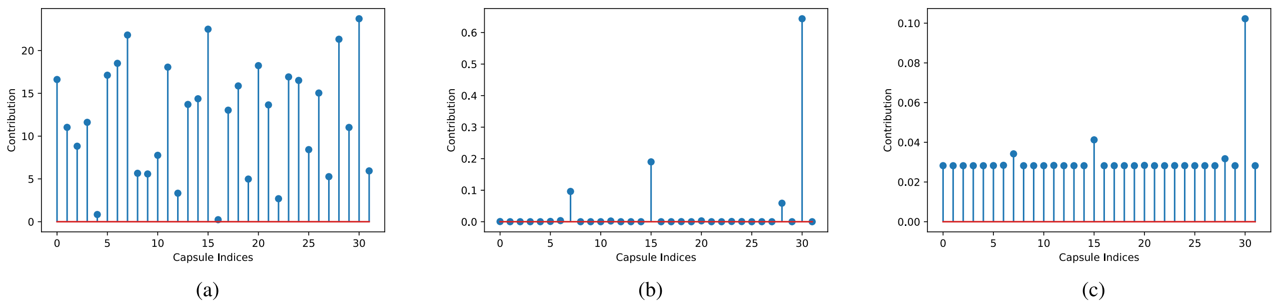

FIGURE 4. Two-round softmax function as leaky-softmax implementation with $\alpha = 2$.

concrete instantiation parameters of higher-level entities. As Hinton *et al.* [4] pointed out, this cluster process can be characterized as non-trivial task since each higher-level capsule receives as input different data points through the multiplication with discriminative transformation matrices. In addition, Hinton *et al.* [4] argued that this situation leads to symmetry breaks and fast convergence. In general, both of these properties are desirable in the context of neural networks. The EM routing procedure obtains as input the votes V and activations $\mathbf{a}$ of all capsules from layer L to capsule j from layer $(L + 1)$. The term *vote* is here used as synonym for a prediction from a lower-level capsule. EM routing processes predictions in vector form.

The routing process starts with the initialization of the responsibilities R_{ij} which a higher-level capsule j takes for a lower-level capsule i. The *inverse temperature* schedule λ provides a specific value for each iteration in the EM process [4]. Both constants β_u and β_a, as well as the inverse temperature schedule λ, are jointly trainable with the entire CapsNet. The temperature schedule can be understood as an implementation of *simulated annealing* [29], where temperature steers the degree of freedom to allow seemingly suboptimal solutions within an optimization process. Usually, temperature starts high and gets progressively decreased during an optimization process to first overcome local optima and, finally, stabilizing the found solution [27]. Theoretically, also again increasing temperature could result in positive effects for solution finding, for instance, if oscillating behavior can be observed. Equally to dynamic routing, it is recommended to run EM routing with three iterations [4].

One similarity between EM routing and the enhanced dynamic routing variant denotes the use of capsule activations to influence the connection strengths between lower and higher-level capsules. The main difference between both routing procedures represents the high-computational effort within EM routing compared to dynamic routing, since EM routing involves more complex calculations. However, the cluster finding approach with Gaussian mixtures in EM routing appears more powerful than the trivial dynamic routing with dot product agreement and, therefore, EM routing harbors potential for faster convergence during model training.

```
procedure k-Means Routing($\hat{\mathbf{U}}, r$)
    Initialize $\mathbf{v}_j \leftarrow \frac{1}{k}\sum_i \hat{\mathbf{u}}_{j|i}$
    for $r$ iterations do
        $b_{ij} \leftarrow \langle \frac{\hat{\mathbf{u}}_{j|i}}{||\hat{\mathbf{u}}_{j|i}||}, \frac{\mathbf{v}_j}{||\mathbf{v}_j||} \rangle$
        $c_{ij} \leftarrow \underset{j}{softmax}(b_{ij})$
        $\mathbf{v}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
    return $squash(\mathbf{v}_j)$
```

LISTING 3. Pseudo code of the k-Means routing procedure [11]. The original term $W_{ij}\mathbf{u}_i$ was replaced with $\hat{\mathbf{u}}_{j|i}$ since incoming predictions are already transformed inputs; Notation adjusted for listing uniformity in this paper.

In coherence with EM routing, **k-Means routing** assigns incoming predictions to clusters represented by higher-level capsules. k-Means routing is based on the prominent *k-Means* [30], [31] (also called *Isodata* [27]) clustering algorithm. In the k-Means algorithm, a cluster k is represented as the arithmetic mean μ_k (often named as *prototype*) of its associated data points [28]. During the iterative fitting process, data points are alternating re-assigned to their nearest cluster centre and the means of clusters are re-calculated until a termination criterion is satisfied [28].

The k-Means routing procedure, illustrated in Listing 3, follows the same intuition like EM routing. In that sense, higher-level capsules correspond to clusters and predictions of lower-level capsules represent data points. The procedure begins with the initialization of each cluster prototype $\mathbf{v}_j$. For this purpose, a mean is calculated over the passed vector-shaped predictions $\hat{\mathbf{U}}$. Afterwards, the clustering process is applied for r iterations to repeatedly determine the coupling coefficients c_{ij}. As measure for the agreement between cluster prototypes and predictions, the cosine-similarity b_{ij} is used. Finally, each prototype $\mathbf{v}_j$ gets processed by a squash function and, then, returned as instantiation parameters for higher-level capsules. As squash function, the same variant which was used in dynamic routing can be utilized.

One essential difference of k-Means routing compared to conventional k-Means clustering represents the use of *soft assignments* instead of *hard assignments* [28].

Hard assignments mean that each data point is exactly assigned to one cluster. Since k-Means routing applies the softmax function on the measured similarities, probability values for cluster memberships are determined and not only the nearest cluster gets assigned to a data point. According to Ren *et al.* [11], if for the number of iterations r holds $r \rightarrow +\infty$ then k-Means routing tends to produce only hard assignments, which could be problematic because a lower-level feature is normally part of more than one higher-level feature. To mitigate this undesirable behavior they also used $r = 3$ iterations in their experiments [11].

The k-Means clustering algorithm can be characterized as a special implementation of the general EM algorithm [28]. So, k-Means routing is also strongly related to EM routing. This holds in particular since k-Means routing realizes as well soft assignments. The main benefit of k-Means routing compared to EM routing is its computational simplicity [27]. Besides, EM clustering generally needs many more iterations to converge to an approximate level as k-Means with the same number of iterations [28]. However, EM routing has the ability to form Gaussian clusters whereas k-Means routing can solely consider spherical clusters. With respect to the computational effort, k-Means routing is comparable to dynamic routing. Furthermore, both routing procedures conduct a squash function to normalize instantiation parameter vectors. One main difference between k-Means and dynamic routing is that dynamic routing repetitively performs the squash function, whereas k-Means routing only applies it once at the end of the routing process.

To be precise, **static routing** is not actually a realization of routing-by-agreement since it propagates capsule outputs from a preceded layer directly to the subsequent layer and does not follow any agreement approach between lower and higher-level capsules. The motivation behind involving such a contrary approach in the considerations of this paper is to additionally evaluate the routing-by-agreement concept in a general manner for text classification. This is especially useful for assessing the high computational effort which normally arises with the use of iterative routing-by-agreement approaches.

Listing 4 illustrates the pseudo code for a simple static routing procedure. This procedure only adds up all incoming predictions from the previous layer and normalizes the resulting sum with a squash function. The application of a nonlinearity like a squash function is here stringently required, since nonlinearities within neural networks are essential for their learning capacity. In a conventional neural network nonlinearities are implemented using activation functions. Actually, capsule networks can provide two types of nonlinearities.

The first represents the specific inter-layer communication (iff any nonlinearity is included, e.g. routing-by-agreement) and the second can be realized based on auxiliary normalization or activation functions. Kim *et al.* [10], the authors of static routing, assumed that in the text domain examining exact spatial relationships could be counter-productive because of the inherent variability in text. For instance, they argued that the precise order of sentences is not a crucial feature for the correct classification of a document [10].

```
procedure Static Routing($\hat{\mathbf{U}}$)
    $\forall j \in \Omega_{L+1} : s_j = \sum_i \hat{\mathbf{u}}_{j|i}$
    return $squash(\mathbf{s}_j)$
```

LISTING 4. **Pseudo code for a static routing procedure, based on the concepts from [10].**

The major benefit of static routing embodies its marginally computational effort compared to all previously presented routing procedures. Another advantage could be that all lower-level features are propagated to all higher-level capsules which leads to a fair treatment for all capsules. On the other hand, this also supports the transmission of noise. Static routing integrates no attention mechanism, whereby selective power gets removed from a CapsNet.

If routing procedures are technically realized with an uniform interface, routing algorithms can be easily replaced in a CapsNet architecture and are freely selectable per network layer. The possibility for choosing different routing procedures per layer, offers great potential for exploiting domain-specific knowledge in distinct abstraction levels. Evidently, trainable parameters in routing algorithms are also adaptable per CapsNet layer.

B. TEXT CLASSIFICATION BASICS

The task of text classification belongs to the large field of *Natural Language Processing (NLP)* [25], [32], [33] in which spoken or written text gets processed by a computer system to understand or generate natural language. Text can be grasped as times series data that comprise observations in the progression of time. In this view, observations correspond to characters or words, and the progression of time represents the semantic accumulation defined by the sequential order of characters or words. In general, time series data can be stochastically described by a *stationary* or *nonstationary* distribution [28]. To act on a stationary distribution means that certain statistical attributes (e.g. mean or variance) of time series data are invariant in time [34]. The task of recognizing patterns in natural language can be categorized as investigation of a stationary distribution, since natural language follows certain structures like grammar or other semantics that are preserved over time. Evidently, a statistical model can encounter previously unknown vocabulary which can influence the underlying distribution, but such exceptions can be handled in various ways.

1) LANGUAGE MODELING

First of all, it is necessary to internalize the inherent characteristics of natural languages and the difficulties that come along with them. Words in natural languages can be assigned to *lexical categories* which can be in turn summarized in *syntactic categories* and, finally, joined to form specific *phrase structures* of sentences [33].

The exemplary sentences in Fig. 5 emphasize the variability in natural languages and depict the complexity for a computer system to capture semantics. All three sentences consist of the same four words and follow the same sentence structure of *subject-verb-object*. In addition, the adjective "hot" varies in its position from sentence to sentence, and in the last sentence the punctuation was also changed. Despite the nearly identical looking of the sentences, their meanings could be quite different. For example, if we regard the expressed sentiments within each sentence, the first sentence may be the response to the simple question: "Would you like to have some tea?". In this situation, no relevant sentiments would be present. But if a waiter serves you a cup of cold coffee, you may complain about the temperature of your coffee. This could be expressed with the second sentence because an adjective at the end of a statement often signals an intonation. A correlated sentiment with such a sentence can be displeasure. The latter sentence even goes a step further and reveals the strong emotion of anger which is highlighted by the use of a comma and an exclamation point.

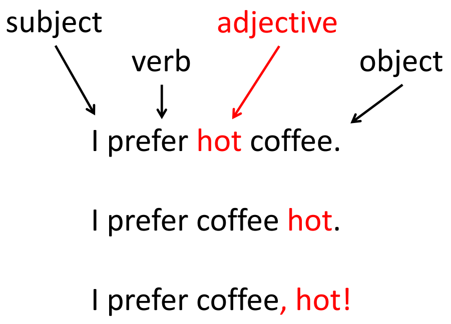

FIGURE 5. **Three exemplary sentences with similar meaning to visualize the variability in natural language. In particular, the position of the adjective in combination with punctuation has a significant impact on the sentiments, expressed by these sentences.**

Another important aspect, which is visualized in Fig. 5, is the task-specific assignment of lexical categories. In other tasks than sentiment analysis, it could be irrelevant to detect subjects and objects in sentences. So, the description *object* for the term "coffee" could be replaced with the lexical category of *noun* and the label *subject* for "I" could also be substituted with the grammatical class *pronoun*. Hence, this trivial example already gives a great outlook about the inherent complexity in natural language. One can imagine how this complexity grows with the use of nested and sophisticated sentence structures.

However, changing word order and punctuation does not reflect the only possibility for affecting the expressiveness of statements in natural language. A look at human interaction in a more general manner, reveals that semantics in natural language are transported over various *communication channels*. Therefore, to fully understand subtle intentions and sentiments in statements requires the consideration of all these channels. As Fig. 6 depicts, at least five communication channels can be identified within human interaction.

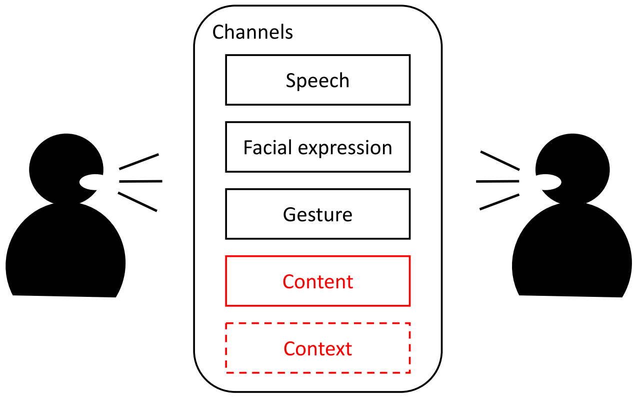

FIGURE 6. **Different communication channels in human interaction. In textual data, the content of statements is fully observable and subtle context can be partially inferred through content accumulation.**

Undoubtedly, transferring semantics in human interaction involves many cognitive abilities. For instance, the intonation of spoken words or breaks between two words enrich the content about what was said with semantics. Other communication channels represent facial expression and gesture that can steer attention and, evidently, transport emotions to a conversational partner.

Besides, each content can be encapsulated within a context or situation. Consider the following sentence: "I never liked John.". Without any context, one could say it is more or less an emotionless statement, reflecting an opinion. But if we know that John recently died, the above sentence could seem heartless or just inappropriate. Another point is that words can also be ambiguous in their meaning, for example, the word "duck" could mean a waterfowl or describing a movement depending on the concrete context [33]. Principally, a content can have long, mid or short-term dependence on a context. In the case of text classification, a computer system can fully access the communication channel *content* based on the provided textual data. Generally, computer systems can build up a context based on word and sentence sequences. In some cases, further context information can also be supplied from the given application domain and appended meta-data (e.g. keywords or topic assignments) to the text data. All other channels are completely inaccessible with respect to the task of text classification. Thus, a lot of semantic power gets lost and the severity for the task of text classification increases.

2) REPRESENTATION LEARNING

Both aforementioned capsule models offer a high expressiveness for considered entities through their distributed representation. The key idea behind distributed representations lies in their ability to implicitly learn diverse concepts and decouple them from each other. So, each concept can be interpreted as a degree of freedom in the entity space. Each instantiation parameter can represent an unique concept. An important characteristic about distributed representations denotes their capability to learn gradual changes in concepts without requiring training samples for each possible configuration. This property enables strong generative power for a representation learning model by creating a continuous entity space, with the concepts as different directions. For instance, a concept in the image domain could be a color scale or skewness degree. In particular, Sabour *et al.* [3] observed that instantiation parameters can encode compositions of features like "width and translation" or "scale and thickness" for the visual classification of handwritten digits. One concept can also signal that a certain object is available in an image or not. Obviously, in this case emerges a lack of explainability about intermediate states between the endpoints existent and non-existent, but since the model learns this concept implicitly, in general one cannot influence the behavior for gradual changes. [25]

To optimally support the high capability of capsules to describe entity properties, a variety of task-specific and expressive features must be provided as input to the initial capsules. Based on the inherent complexity in natural languages regarding word order, sentence composition, textual padding and sequential behavior, which determine the text meaning as much as the concrete words, it is essential to choose proper representation learning models for a text classification task. One way to distinguish between representation learning models is to consider statistical measures, spatial relations, sequential behavior and relevance filtering for accessing textual patterns. In that sense, representation learning models can be differentiated into **statistics-, structure-, sequence-** and **attention-based** models (cf. [12]). Additionally, **word embeddings** can be categorized as a representation learning model for expressing relationships between words, or more generally tokens [25].

Statistics-based models compute statistical measures for describing textual data. Usually, such models use a sparse *Bag-of-X* [35] representation which consists of a collection of X-like elements and their corresponding number of appearance (cf. [12]). One simple variant of this representation is the *Bag-of-Words (BoW)* representation which separates a textual sample in individual words and counts the occurrence of each word [33], [35], [36]. Evidently, for providing comparability between BoW representations of two textual samples, a common vocabulary must be predefined (e.g. using the training data). Since the vocabulary has a fixed size v, each textual sample can be described by an integer-based vector with length v. Using a BoW representation with an extensive vocabulary results in very sparse vectors. The weighting of terms in textual samples on the basis of word counts is referred to as *term frequency (tf)* [36]. As a consequence of using tf, the spatial structure of textual samples gets completely lost, which removes a lot of semantics [35]. In particular, relevant words for classifying a textual sample occur much more rarely than other words such as articles, pronouns, prepositions or conjunctions. Seemingly irrelevant words for a given task are called *stop words* [35], [36]. One strategy for mitigating the noise ratio in BoW representations is to reduce the common vocabulary by removing such stop words. However, the trend in information retrieval systems shows that the list of stop words generally shrinks [36].

Another possibility depicts the use of a *Bag-of-n-Grams (BonG)* representation [35]. An *n-gram* corresponds to a sequence of tokens with length n [25], [35]. n-Grams with values $n \in \{1, 2, 3\}$ are called *unigram*, *bigram* and *trigram*, respectively [25]. In general, the type of tokens can be arbitrary, for instance, tokens may denote words, syllables or characters [33]. Fig. 7 illustrates a possible way for extracting trigrams with an unidirectional sliding window approach. Since no padding is used, the sliding window starts at position 1 and moves until it reaches position 4 to ensure valid trigrams. With an increasing window size n the ratio of semantic preservation grows. But this structure preservation claims auxiliary costs for storing and processing n-grams since high values of n result in much more tokens per original text fragment [35]. Using n-grams as smallest operating units with larger values for n (means $n \geq 2$) leads to a combination of statistics- and structure-based models. In some sense also a notion of sequential behavior is retained since an n-gram summarizes n tokens in a predefined order.

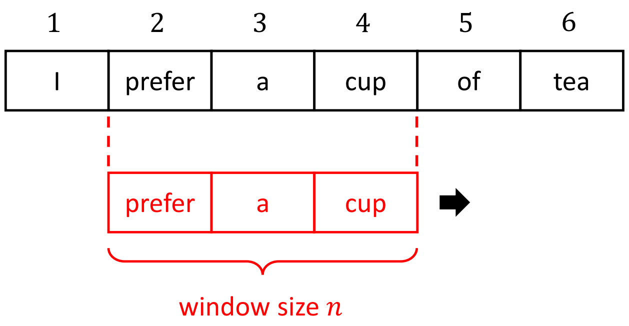

FIGURE 7. Extraction of n-grams with a sliding window approach ($n = 3$).

Structure-based representation models try to retain spatial information of textual data. Spatial information about text is especially relevant in situations where an understanding of complex semantics is essential to fulfill a classification task. One example for such a task could be sentiment analysis since feelings are often subtly encoded in certain grammar structures. One way for capturing spatial relations between tokens was already presented with the concept of n-grams. Regarding n-grams, the size of n determines the degree of representing structural information.

Retaining structural information of textual data does not stringently require to store information about directly adjacent words. It rather means that relevant spatial information is preserved for solving a certain classification task. Following the simple sliding window approach in Fig. 7 for n-gram extraction, a straightforward realization in neural networks can be established with a convolutional layer. Specifically, a convolutional operation can be extended by a *dilation* rate which defines the expansion of a convolutional filter with respect to the distance between filter elements in the considered receptive field [37]. Yu *et al.* [37], for example, utilized convolutional layers with various dilation rates to achieve *multi-scale context aggregation* for dense prediction in the image domain. Zheng *et al.* [12] correctly pointed out that different dilation rates in the extraction of word-based n-grams lead to varying granularity levels which could describe *word-, phrase-, clause-* or *sentence-level* structures.

CapsNets are inherently able to describe spatial relationships through their distributed representation and the use of transformation matrices [2]–[4]. Moreover, a routing-by-agreement procedure, which tries to accomplish part-to-whole assignments between lower and higher-level entities, can effectively support the learning of spatial features [3], [4]. Since spatial relations can represent crucial properties depending on the concrete task, it should be quite natural that capsules are forced to learn this structural context. In coherence with this, Wu *et al.* [17] stated that capsules could depict *grammatical structure information* and *spatial distance of local features*.

In contrast to structure-based models, **sequence-based** models do not directly focus on a special contextual structure but rather accumulate over text sequences to memorize relevant text fragments and to describe various-term dependencies. Fig. 8 conveys the intuition behind forming a context based on previous inputs. In this example the current observed element represents the fifth word ''of''. The sequence-based model manages a context, visualized with a color code. The color code symbolizes the ratio of stored context information from previous words. This context can then get involved into the consideration of a new element.

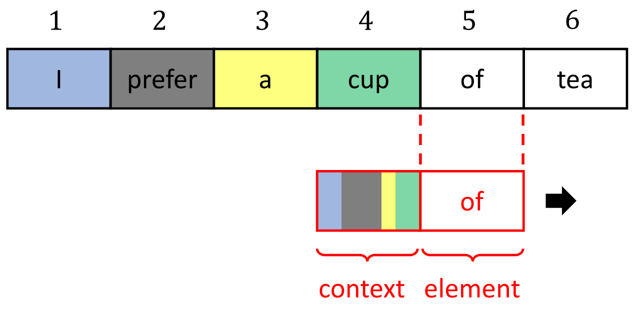

FIGURE 8. **Visualization of context accumulation with unidirectional processing of word sequences.**

Fig. 9 shows a recurrent neuron which receives as second input per time step t its previous output y_{t-1}. The terms x_t and y_t correspond to the input x and output y at time t. A mathematical definition [25] for the output of a recurrent unit is

$$y_t = f(x_t, y_{t-1}; \theta). \tag{7}$$

Thus for computing the neural output y_t, a function f with a defined set of parameters θ consumes the current input x_t and the accumulated neuron output y_{t-1}. There exist various ways for the concrete implementation of function f. The specification of function f strongly depends on the question about which contextual information is needed for solving a certain task. A *Recurrent Neural Network (RNN)* [38] represents a neural network that is composed of recurrent neurons or contains in some way backwards directed connections [25], [32]. In opposite to convolutional layers which share parameters through the repeated application of the same kernels on sequences of samples, recurrent layers share parameters by repetitive accumulating previous samples using the same function parameters [25], [38].

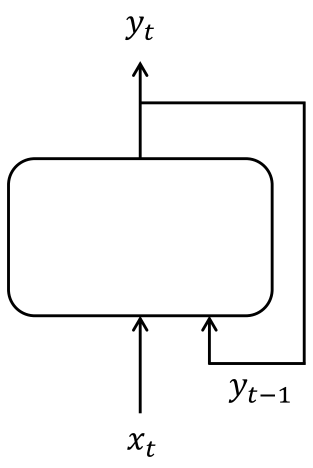

FIGURE 9. **Basic recurrent neuron, graphic inspired by [32].**

One major challenge in sequence-based models denotes the ability to represent long-term dependencies. Despite the great potential of RNNs to learn contextual information about sequential data, the difficulty of capturing long-term dependencies grows with the total sample length. This central problem arises from the mostly applied optimization algorithm *backpropagation* [38] which enables learning within neural networks, but suffers in *deeper* neural nets from the effects of *exploding* or *vanishing gradients* [39]–[41]. Since RNNs can be easily transformed into feedforward neural networks with shared weights by unfolding in time, this often yields to very deep net structures [38], [39]. For mitigating the negative effects of exploding or vanishing gradients, adjusted learning algorithms [39] may be applied instead of traditional backpropagation, initialization of weights in a RNN can be done using prior knowledge [39], or normalization strategies like *Batch Normalization (BN)* [40] or *Layer Normalization (LN)* [41] can be utilized. Roughly speaking, normalization strategies standardize activities in neural network layers to have zero-mean and unit-variance for preventing *internal covariate shift* to the subsequent layer, which significantly speeds up training convergence [40], [41]. Especially, LN has a strong positive impact on stabilizing the learning process within RNNs since it can handle variable-length inputs by standardizing layer outputs individually for each sample and each time step [41]. However, BN provides better statistic estimates, if fixed-length inputs are given, by considering mini-batches instead of single samples [40].

Apart from simple recurrent neurons, other important and specialized cells for storing contextual information about sequential data are *Long Short-Term Memory (LSTM)* [42] and *Gated Recurrent Unit (GRU)* [43]. Both LSTM and GRU belong to the class of *gated RNNs* that are able to save contextual information regarding different time spans, can learn to filter relevant information by just forgetting irrelevant parts of sequences, and should provide stable derivatives to prevent the exploding and vanishing gradient problem [25].

A neural gate can be realized as multiplication operation between a weighted input wx_t and a control signal g_t, which equips a neural network architecture with the opportunity to propagate only relevant signals [42]. An elementary neural gate is illustrated in Fig. 10. The multiplication operation is displayed as $\otimes$ operator that consumes as inputs wx_t and g_t. Since the control signal g_t results from an arbitrary input, processed by the sigmoid function, values of g_t lie in the range of [0; 1]. This enables a neural gate to let pass any percentage of an input signal wx_t. A crucial property of such a neural gate represents the ability to steer error flows by learning when to interrupt error flows and when to pass them [42]. This control mechanism can be mathematically described with the formulae

$$y_t = (x_t w) g_t; \quad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial y_t}\frac{\partial y_t}{\partial w} \tag{8}$$

$$\lim_{g_t \to 0} \frac{\partial y_t}{\partial w} = \lim_{g_t \to 0} x_t g_t = 0; \quad \lim_{g_t \to 1} \frac{\partial y_t}{\partial w} = x_t. \tag{9}$$

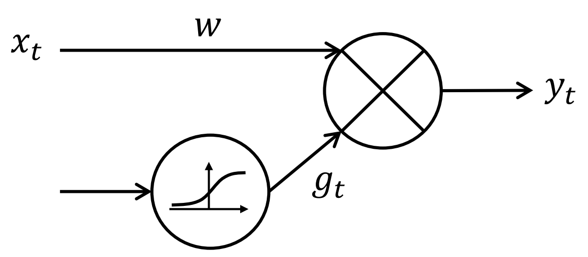

FIGURE 10. **Neural gate with sigmoidal control input g_t, inspired by [25], [32], [42].**

Equation (8) illustrates the backpropagation of a computed error from a loss function L with respect to the parameter w. Since $\frac{\partial y_t}{\partial w} = x_t g_t$, the gate signal g_t controls the error flow. Equation (9) states both edge cases with a completely closed ($g_t \to 0$) and open ($g_t \to 1$) gate.

A regular LSTM cell comprises a *forget*, *input* and *output* gate, which steer the deletion of the previous state, the passing of external inputs and the releasing of cell's output signal, in the given order [25]. All gates decide to open or close based on the previous state and the current input [25], [32], [42]. The input gate partially solves the problem of dealing with long-term dependencies by protecting the accumulation of contextual information from irrelevant inputs [42]. This significantly increases the maximally possible time for grasping past context. Moreover, the output gate can protect subsequent units from perturbed, intermediate states [42].

In general, GRU cells are considered as much easier to compute and implement compared to LSTM cells. Contrary to LSTM, GRU cells solely contain two gates, namely a *reset* and *update* gate. Similar to the forget gate in LSTM, the reset gate can be used to delete information of the current state. This happens in coordination with the update gate, since it determines the ratio of the previous state which is kept for the next state. [43]

It is important to note that the behavior of neural gates within both LSTM and GRU cells is learnable during model training, with the use of weights for the computation of a control signal g_t [25], [32], [42], [43]. A LSTM or GRU cell can represent function f in (7) [43]. It is difficult to meaningfully distinguish LSTM and GRU from each other, since both cells act very similar. However, both types of gated RNN cells, more or less, solve the problem of various-term dependencies and mitigate the flaw of effectively describing long-term dependencies. This offers great potential for a computer system to *understand* semantics in natural language.

Attention-based models can learn to concentrate on specific parts in samples that are necessary for solving a given task [32]. In the text domain, attention mechanisms mainly have their origin in the task of neural machine translation which tries to transform textual data into equally meaning textual data in a different language [44], [45]. Bahdanau *et al.* [44] proposed a neural machine translation model including a jointly trained feedforward neural network for determining energy values for words at time step t, then the energy values were converted into valid probabilities and finally multiplied with the features for the corresponding words in order to represent attention on specific words. This proceeding can be abstracted to a simple attention mechanism, as visualized in Fig. 11.

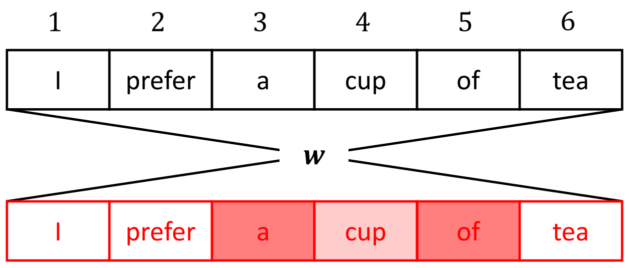

FIGURE 11. **Attention mechanism for weighting a word sequence through element-wise multiplication with a learned weight vector w. The color scale from white to red signals the focus on individual words from high to low, respectively. The constraint $\sum_i w_i = 1$ holds.**

The exemplary word sequence gets element-wise multiplied with a learned weight vector $\mathbf{w}$. Similar to the work of Bahdanau *et al.* [44], the elements of the weight vector should sum up to one, to ensure a valid probability distribution. In this example, applying the weight vector $\mathbf{w}$ leads to concentrating on words such as "I", "prefer" and "tea" whereas "cup" is not that much important, and the words "a" and "of" could be completely ignored. The idea of focusing only on relevant parts of input data for solving a certain task is as simple as powerful. It significantly diminishes noise in input data and behaves similar to a dimensionality reduction method. This property is particularly desirable for the processing of long sentences [44], [45], or more generally speaking for long text sequences. Principally, a variety of attention mechanism realizations are possible. For instance, Luong *et al.* [45] experimentally compared a *global* attention implementation with a *local* one (that restricted attention to a defined window) for the task of neural machine translation. Especially, they explored different methods for computing attentional scores such as the dot product, multiplication with an intermediate weight matrix and weighted concatenation of hidden state vectors [45].

The use of routing-by-agreement in CapsNets inherently realizes a kind of *parallel attention mechanism* [3], [8] between consecutive layers, by weighting contributions from lower-level capsules to higher-level ones. Thus, routing-by-agreement can be viewed as a special type of an attention-based representation model. Similar to the attentional implementation of Bahdanau *et al.* [44] and Luong *et al.* [45], CapsNets with routing-by-agreement allow a model to implicitly learn to concentrate on relevant aspects in input signals.

Word embeddings address the problem of ignoring relationships between words, as it occurs using e.g. BoW representation. For this purpose, word embeddings provide a distributed representation that has the ability to describe latent relations between individual words by their location in the word embedding space [25]. Latent relations in natural languages are composed of *syntactic* and *semantic regularities* [46]. More precisely, a distributed and real-valued representation enables the encoding of contextual features and offers a way for expressing similarities between words, which both should improve the generalizability of a language model [25], [46], [47].

Various strategies are imaginable for transforming word indices or one-hot encodings into an expressive and distributed representation. One strategy is the use of a simple RNN which projects an incoming word, then processes the result with a simple RNN unit and finally outputs a probability distribution over the total vocabulary [46]. Another common strategy represents the method *word2vec*, proposed by Mikolov *et al.* [48], which generates word vectors through extracting the context of the regarded word. For generating word embeddings with the word2vec framework, Mikolov *et al.* [48] designed the two models *continuous BoW* and *continuous skip-gram*. Both models extract contextual information of a word w through a learned projection of or for the surrounding words of w [48].

Besides word2vec, a currently state-of-the-art technique for computing expressive word embeddings represents *Global Vectors for Word Representation (GloVe)* [47]. Similar to word2vec, GloVe creates word representations based on word co-occurrence statistics within the training corpus. Instead of training a model using local context windows, as e.g. in the continuous skip-gram model, GloVe's authors claim their method to have a more global view on syntactic and semantic regularities by considering global statistics of the training data. In fact, GloVe significantly outperforms word2vec in capturing syntactic and semantic regularities. [47]

Other approaches for creating word embeddings operate on subword level to represent words as compounds of word parts [49]. The general idea is related using n-grams on the word level, but subword embeddings ensure deeper insights in language characteristics like word endings and the meaning of combined words, that is especially advantageous in natural languages like German (e.g. see the German word "Blumentopferde" which means "potting compost" in English) [49].

Fig. 12 illustrates two ways of how the use of word embeddings can reflect task-specific or semantic relationships between words, solely by their learned location within a word embedding space. Subfigure (a) depicts the implicit forming of word clouds in the word embedding space. In this example, each cloud can be assigned to a task-specific topic like *Past & Future* or *Middle Ages & Technology*. Therefore, exploring trainable word embeddings has the potential for knowledge gaining about a considered problem domain. Subfigure (b) shows word embeddings which were learned with focus on capturing syntactic and semantic regularities, as proposed in [46]–[48], [50]. Ideally, learned contextual properties can be expressed with *vector offsets* between word embeddings [46]. For instance, subfigure (b) provides a construction of the distributed representation of the word "queen" based on the word "king", by substituting the property "man" for "woman". This construction can be mathematically applied by a subtraction followed by an addition, as stated in subfigure (b). This cryptic formulation elucidates the circumstance that *king* behaves to *queen* like *man* behaves to *woman*, since the difference in both pairs lies in the gender. Evidently, in practice the precision of word embeddings and their relations are limited. Hence, an appropriate similarity measure for vector spaces must be applied for determining the nearest word embedding for a computed vector [46].

As a remark, known *idiomatic phrases* like "New York", "Air Canada", "Boston Globe" etc., should be identified and considered as one word in an embedding space, to deal with their special meanings [50]. One crucial benefit of word embeddings is that pretrained word representations can be exploited for other related tasks [46]. Word embeddings are in strong coherence with CapsNets because both techniques build expressive, distributed representations of observed entities.

3) TRENDS IN TEXT CLASSIFICATION

Since the NLP domain consists of a wide spectrum of different classification tasks including sentiment analysis, content summarization, aspect extraction, information retrieval, adequacy check etc., a tremendous number of methods were developed and exploited to solve specific text classification tasks. To keep things short, some selected methods for text classification are presented in the following. The aim is to

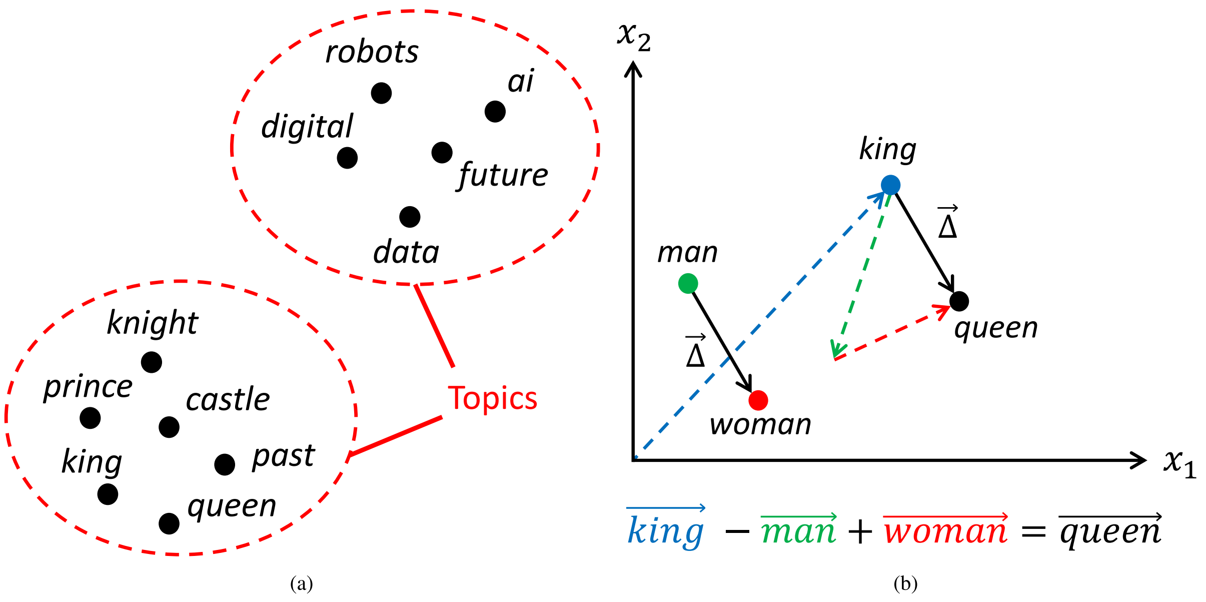

FIGURE 12. (a) Visualization of learned relationships in a word embedding space, based on [25]. (b) Exemplary construction of a similar word by manipulating specific properties of a word embedding, based on [46], [48], [50].

explain the progress in time of text classification technologies and to identify trend-setting developments in this area.

The first attempts for accessing patterns in textual data have their origin in statistical considerations of term frequencies with respect to single documents and relative to all documents [51]. For example, the statistical method *term frequency – inverse document frequency (tf-idf)* ranks documents by the relevance of their containing terms and is mostly used in the area of information retrieval [36]. However, statistical measures like tf, inverse document frequency (idf) or tf-idf can be principally utilized for arbitrary tasks in the text domain. A contemporary statistics-based approach for text classification, which achieves for many datasets comparable results to extensive neural networks, in much less time, represents the combination of BoW or BonG with word embeddings and a simple linear classifier [52]. Exemplary linear classifiers for text classification are *Perceptron-Like Classifiers* [53], *Naive Bayes* [54] and *Support Vector Machines (SVM)s* [55], [56]. (Of course, SVMs can be easily transformed into nonlinear classifiers with the aid of an appropriate kernel function [56].)

One can recognize a trend in text classification methods in the direction of neural network technologies. Neural network approaches include neural-based word embeddings [46]–[48], RNNs [38], [57]–[59], Convolutional Neural Networks (CNN)s [58], [60]–[62] and other net structures. CNNs offer the benefit of processing variable-length input using local feature extractors with shared parameters [60]. Text classification with CNNs is mostly applied on word-level, but character-level models are also common [61], [62]. Despite CNNs are not as much prominent for context building as RNNs regarding NLP, stacked CNN architectures supply also capability for extracting underlying context. The most prominent recurrent cells represent LSTM [42] and GRU [43]. Although RNNs can access sequential patterns, they still suffer from long-term dependencies [39]. Dai *et al.* [63] proposed one feasible solution to mitigate the flaws of LSTM for describing long-term dependencies by initializing a LSTM-based RNN with weights obtained from a pretrained autoencoder. An extension to a regular RNN represents a *Bidirectional RNN (BRNN)* which manages contextual information for the past and future in sequential data [64]. Evidently, bidirectional processing can also be exploited for other neural network technologies. Often, diverse neural network approaches are combined to create more powerful feature extractors, or to improve classification ability. For instance, Wang *et al.* [58] extracted local features with two parallel arranged CNN layers and, afterwards, merged all local features to obtain the overall input to a subsequent LSTM layer.

Another trend regarding text classification models denotes the enrichment of existing models with attention mechanisms to significantly improve resulting performance. Yang *et al.* [65] conceptualized a hierarchical attention network for document classification which exploits document structure by step-wise building a comprehensive context, beginning with words up to sentences. Hu *et al.* [57] designed a model consisting of stacked LSTM hidden states aggregated into an attentional hidden state vector. Vaswani *et al.* [66] elucidated the power of attention mechanisms in the text domain by proposing a transformer architecture for neural machine translation that refrains from the use of RNN and CNN layers, by representing a system solely composed of *self-attention* modules in combination with *multi-head attention*.

Interestingly, CapsNets reflect this progress in the area of text classification by embodying a mixture of neural network technologies with a distributed parallel attention mechanism. Utilizing word embeddings in combination with CapsNets is also strongly consistent due to the use of distributed representations in both techniques. Major advantages of distributed representations and parallel attention mechanisms are noise and dimensionality reduction (through focus and robust representation learning) and, eventually, diminished training time. Moreover, robust representations often decrease the number of required training data. Thus, it can be assumed that in near future neural network technologies, attention mechanisms, distributed representations and especially CapsNets will still dominate the field of text classification.

III. USED TEXT DATASETS

In order to obtain convincing results in the analysis part of this paper, six text classification datasets are regarded with a wide spectrum of characteristics. The six datasets are chosen to satisfy the following criteria:

- small and large-sized datasets
- varying number of classes
- diverse classification tasks
- different sample granularity levels.

The size of a dataset contributes to the overall task complexity, more data have the potential to embody more latent variables, but neural networks are known as technology which needs a lot of data to generalize well. Specifically, large datasets prevent overfitting of models, because just memorizing certain terms in samples would not be enough for solving a task. Another indicator for task complexity represents the number of classes. Generally, task severity increases with the amount of classes within a dataset. Intuitively, considering many classes amplifies the division of the training corpus, which reduces the number of samples per class. Furthermore, a high number of classes requires a classification model to capture many latent variables for ensuring class discrimination. This situation gets even exacerbated when a dataset is imbalanced, means classes have different numbers of associated samples. Certain classification tasks are more challenging than others. For example, in sentiment analysis seemingly positive phrases can be embedded in a strongly ironic context which actually represents contrary sentiments. In addition, some classification tasks mainly depend on syntactic and sequential structures, whereas others can solely be solved by observing relevant terms. The last criterion for dataset selection refers to the granularity level of samples. Granularity levels may be *document, sentence, phrase, statement* etc. and provide a simple measure for determining task difficulty. For instance, a dataset with samples that represent whole documents have the potential to suffer from the well-known *curse of dimensionality*. With respect to sequential data, enlarged sample granularity levels particularly cause negative effects for grasping long-term dependencies. The remainder of this section introduces the chosen datasets and discusses significant properties derived from their statistical values.

The **Subjectivity Dataset (SD)**[1] provides a collection for learning the distinction between *objective* and *subjective* statements. Each sample statement corresponds to a sentence or textual, sentence-like snippet. All objective samples were gathered from *Internet Movie Database (IMDB)*,[2] whereas all subjective ones were compound of customer movie reviews from *Rotten Tomatoes*.[3] Sentences and textual snippets contain at least ten words. The number of samples per class is strongly balanced. But the dataset can marginally comprise wrong labelling, since class assignments to samples were not manually verified. [67]

LIAR[4] serves as benchmark dataset for detecting fake news. This dataset contains short statements which were manually labeled by *PolitiFact*.[5] Additionally, labelling decisions are supported with comprehensive resources like analysis reports or links to source documents. This dataset is divided into the six classes: *pants-fire, false, barely-true, half-true, mostly-true* and *true* where the truth content continuously increases in the given order. Besides the short statements, the dataset provides auxiliary meta-information about surrounding context and the associated speaker. For instance, context information concerns the subject, venue and used media. Speaker information among others considers knowledge about the truth content of previous statements, the resident state and the belonging party. The LIAR dataset has a little imbalance regarding the class *pants-fire* since this category has approximately half as many samples as the other classes. [68]

In this paper, no meta-information supplied by the LIAR dataset is used for text classification. This means that no additional context or speaker information provided by the dataset is fed into the analysis model. The idea behind this proceeding is to examine if CapsNets are able to find latent variables in syntax or certain words to detect fake news. In principle that should be possible since CapsNets offer more representational capabilities than regular neural networks.

The **20 Newsgroups (20-NG)**[6] dataset is a corpus of email-like conversations in twenty newsgroup forums with different topics. "Email-like" means that each sample contains header information including *From*, *Subject* and *Organization* fields as usual in emails. Furthermore, each email-like sample describes questions or answers of users under the given subject. The number 20 refers to the various topics such as diverse politic talks, computer system issues, scientific exchange etc. It is not exactly clear who created the 20-NG dataset. In this paper, the "bydate" dataset version (some headers and duplicates were removed) from Jason Rennie's

[1]Available at www.cs.cornell.edu/people/pabo/movie-review-data/rotten_imdb.tar.gz

[2]www.imdb.com

[3]www.rottentomatoes.com

[4]Available at https://www.cs.ucsb.edu/~william/data/liar_dataset.zip

[5]www.politifact.com

[6]Available at http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz

home page[7] is used. Rennie assumes that the 20-NG dataset was originally built up by Ken Lang for his paper [69] about a recommender system for online news articles. According to Rennie's dataset description, many of the 20 classes are strongly related, whereas others are highly different. For example, the superior topic about computer systems is composed of five subtopics, whereas the general issue about religion has three subclasses. Hence, superior classes are overrepresented within the 20-NG dataset.

The **Clickbait Dataset (CBD)**[8] defines a two-class collection for clickbait detection in online news articles. For that reason, headlines of news articles from reliable and dubious online resources were gathered. To mitigate the problem of false negatives in the samples of the class *clickbait*, the dataset creators let manually validate their assumption about clickbait articles from volunteers. Headline samples for the class *non-clickbait* were collected from a serious online news article provider which internally applies a review process and uses guidelines for preventing clickbaits. In particular, CBD's creators showed that clickbait headlines follow specific syntactic structures that can be helpful for differentiating serious news articles from dubious ones. [70]

The **IMDB Review Dataset (IMDB-RD)**[9] describes a data collection with informal movie reviews from the IMDB. Maximum 30 reviews per movie were taken. Each movie review corresponds to a written consumer feedback in form of a document. Reviews are exclusively categorized into the two classes *positive* or *negative*. Only reviews with high emotional polarity are utilized, means reviews with strong positive or negative ratings. This dataset is completely balanced. [71]

AG's News Topic Classification Dataset (AG-NTCD)[10] comprises a subset of the original *AG's corpus of news articles*[11] dataset which was collected by Antonio Gullí and consists of more than one million news articles. According to Gullí's home page, his dataset was initially used in the papers [73], [74]. Zhang *et al.* [61] created the AG-NTCD as subset from Gullí's original dataset by solely considering the top four classes using the provided title and description of news articles for classification. The four classes with the highest number of corresponding samples are: *World, Sports, Business* and *Sci/Tech*. In opposite to Zhang *et al.* [61], in this paper the AG-NTCD is further reduced by only taking the sentence-like descriptions of news articles into consideration.

As Chakraborty *et al.* [70] correctly pointed out in their paper about detecting clickbaits, online media (especially informal resources) follow specific syntactic rules (e.g. *over-expressed* and *symbolic* punctuation such as "!!!!", "?!", "***", ":-)", "xD" etc.) and use an extended vocabulary with many abbreviations (e.g. "LOL", "OMG", "ASAP" etc.). Since some of the selected datasets consider informal online resources like the CBD or IMDB-RD, it should be helpful to apply a tokenizer which brings along the capacity for additionally recognizing specific internet vocabulary. For this purpose, the *TweetTokenizer*[12] from the Python natural language toolkit *nltk* is used. In more detail, this tokenizer finds regular words, separates punctuation, detects hash tags (e.g. "#thisisahashtag") and recognizes symbolic structures like smileys or arrows (such as ":-P" or "–>"). This tokenizer is applied for determining the subsequent statistical values for all datasets. Moreover, the tokenizer option for converting all tokens into lowercase is activated.

Table 1 lists the central properties for the six regarded datasets. The collection of datasets covers a wide spectrum of different properties. Almost all displayed datasets have distinct tasks and operating levels. In addition, the number of classes for the classification tasks also shows variability. Evidently, the sample granularity levels of the datasets often correlate with the vocabulary size. The given granularity levels can be sorted from fine to coarse: *Headline* → *Sentence* → *Statement* → *Review* → *Email*. This ordering is reflected by the number of tokens $\tilde{t}_y$, as listed in Table 2. The difference between sentence and statement-level is that statements can comprise more than one sentence, but the overall amount of tokens is still small.

TABLE 1. Primary statistics to all six used datasets. The columns c, $\tilde{s}$ and $\tilde{v}$ denote the number of classes, the number of samples and the vocabulary size, respectively. The tilde sign over s and v means that the values are approximated in scale of thousands.

Dataset	c	$\tilde{s}$	$\tilde{v}$	Level	Task Description
SD	2	$10K$	$23K$	Sentence	Subjectivity Check
LIAR	6	$13K$	$15K$	Statement	Fake News Detection
20-NG	20	$19K$	$210K$	Email	Topic Assignment
CBD	2	$26K$	$25K$	Headline	Clickbait Detection
IMDB-RD	2	$50K$	$150K$	Review	Sentiment Analysis
AG-NTCD	4	$240K$	$79K$	Sentence	Topic Assignment

In particular, Table 2 gives a clue about the sample per class balance in each dataset. SD, IMDB-RD and AG-NTCD appear to be totally balanced since the minimum, maximum and average number of samples per class are equal. In the other three datasets some classes could be under or over-represented which may increase task complexity. The number of tokens per sample also indicates classification task severity. All datasets have a high variance in the number of tokens per sample. The big amount of maximal tokens for the AG-NTCD, despite using sentence-level, can be explained by the circumstance that news article descriptions often use semicolons to enrich remarks.

[7]http://qwone.com/~jason/
[8]Available at https://github.com/bhargaviparanjape/clickbait
[9]Available at https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
[10]Available at *TensorFlow Datasets* [72]
[11]http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html
[12]https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.casual

TABLE 2. Minimum, maximum and average values to the number of samples per class $\tilde{s}_i$ and to the number of tokens per sample $\tilde{t}_i$. Again, all values are approximated whereby numbers marked with K are rounded regarding thousands and other values are rounded regarding their decimals.

Dataset	$\tilde{s}_{min}$	$\tilde{s}_{max}$	$\tilde{s}_{avg}$	$\tilde{t}_{min}$	$\tilde{t}_{max}$	$\tilde{t}_{avg}$
SD	$5K$	$5K$	$5K$	7	120	24
LIAR	$1K$	$3K$	$2K$	2	552	20
20-NG	628	999	942	19	$89K$	400
CBD	$10K$	$16K$	$13K$	1	79	12
IMDB-RD	$25K$	$25K$	$25K$	8	$3K$	278
AG-NTCD	$60K$	$60K$	$60K$	3	225	36

IV. RELATED WORK

In 2018, Zhao *et al.* [8] utilized the CapsNet technology with dynamic routing for text classification. They stated that their work presents the first empirical investigation for text modeling using CapsNets, to the best of their knowledge. One central aspect of their work was the design of the enhanced dynamic routing procedure. They showed that enhanced dynamic routing generally led to higher accuracies for six tested benchmark datasets than the classic dynamic routing procedure. For their experiments, the authors designed a simple CapsNet architecture which was composed of an embedding layer, one convolutional layer, one primary capsule layer, one convolutional capsule layer and, finally, one fully-connected capsule layer for realizing text classification. This basic architecture was used for analyzing the text classification potential of CapsNets in two variants: The first variant solely used the basic architecture and the second one utilized the basic architecture three-times in a concurrent manner with different kernel sizes in the first convolutional layer. The computed class probabilities in the second variant were finally combined with a Capsule Average Pooling (CAP) layer. Since the second variant is able to capture features with various n-gram granularity levels, by varying the convolutional window size, it generally results in higher classification accuracies. Another important aspect in the work of Zhao *et al.* depicts the transfer of a text classification model, trained with single-label data, to a multi-label text classifier for free. Such a transfer is possible because of the high capacity of capsules to learn rich representations of the classification entities supported by an intelligent routing between consecutive capsule layers.

In 2018, Srivastava *et al.* [9] adopted CapsNets for solving the contemporary problem of identifying aggression and toxicity in user comments. They focused their experiments on a minimal use of preprocessing, handling unknown words and coping with transliterations. Their CapsNet architecture was strongly oriented on the aforementioned classification model from Zhao *et al.* [8]. The only differences represent the use of a RNN as first feature extractor layer instead of a convolutional layer and the insertion of a *focal* loss. The RNN layer was introduced because of its ability to deal with time-dependent patterns which experimentally led to better results than a CNN. Additionally, the focal loss should hit the difficulty of imbalanced data since examples of toxic user comments are rare compared to the whole amount of available user comments. Their model accuracy outperformed all other considered benchmark neural network methods. Moreover, their CapsNet model was resilient against *Out-of-Vocabulary (OOV)* words, if the word embedding layer was randomly initialized at the beginning of the training.

In 2018, Kim *et al.* [10] comprehensively investigated the potential of CapsNets in the area of text classification. Their work was restricted on capsules consisting of pose vectors and the use of dynamic routing between consecutive capsule layers. Their CapsNet architecture was composed of an Exponential Linear Unit (ELU) gate layer as filter for relevant inputs without losing spatial information, a convolutional capsule layer to build an internal representation of entities and a fully-connected capsule layer for applying the classification task. The classification results showed that the proposed CapsNet architecture with the use of dynamic routing achieves competitive accuracies on the benchmark datasets. Moreover, the authors argued that the dynamic routing procedure, introduced by Sabour *et al.* [3] for visual object recognition, was not suitable for text classification since information about the precise spatial relation of individual words could reduce the model robustness to varying word positions in sentences or documents. Hence, they proposed the static routing procedure which propagates capsule outputs equally weighted to all capsules in the consecutive layer. Their experiments with static routing on the seven benchmark datasets led to accuracies which outperformed dynamic routing and the considered state-of-the-art neural networks in the most cases.

In 2018, Ren *et al.* [11] combined the concept of *Compositional Coding (CC)* with vector-shaped capsules and introduced a routing-by-agreement procedure based on the simple k-Means clustering algorithm. The basic idea behind CC is to build up word embeddings from few basis vectors which should lead to less parameter consumption without significant performance loss. The authors proposed a basic CapsNet architecture consisting of an introducing CC capsule layer and a mixture of recurrent and capsule layers. The CapsNet was tested on eight text classification datasets and the results were compared with the outcome of two highly-specialized state-of-the-art technologies for text classification. The number of parameters of the considered models were adjusted to each dataset to perform best. The results showed that the CapsNet architecture reaches competitive classification accuracies and has in many cases a fewer number of required parameters.

Although there already exists a lot of other work identifiable in the domain of text classification using CapsNets, the above presented works were covered in more detail because they provide argumentations for distinct routing concepts or serve as fundamental source for text classification with CapsNets. Nevertheless for the sake of completeness, some more related work about text classification with

CapsNets is briefly mentioned below: Zheng *et al.* [12] proposed a sophisticated CapsNet consisting of feature extractors for different granularity levels, vector-shaped capsules and an attentive aggregation layer to force versatile structure representation learning. Aly *et al.* [13] showed that a simple CapsNet architecture is sufficient for hierarchical multi-label classification and achieves best results with deep label hierarchies compared to traditional methods. Fei *et al.* [18] integrated a topic and a capsule module into a neural net architecture to extensively capture textual characteristics of sentiments for multi-label emotion classification. Bhattacharjee [14] utilized CapsNets for successfully overcoming the task of detecting textual clickbaits in social media. Similar to [9], Jain *et al.* [15] combined feature engineering, RNN techniques and attention mechanisms with CapsNets to an ensemble approach that applies sentiment analysis for a question-answering system to find inappropriate user-provided text. Wang *et al.* [16] designed a different capsule model, as proposed in [3] or [4], for sentiment analysis. Their much more complex capsule model encapsulates a *representation, probability* and *reconstruction* module. They used each capsule as a standalone detector for one entity and returned the existence probability and an internal vector representation of the observed entity. The internal vector representation was only used for defining one part of the loss function for training the network. Through an analysis of weights in the applied attention mechanism, the authors were able to rank words by their relevance for classification categories. Chen *et al.* [19] introduced a transfer capsule network for learning sentiment polarity in diverse aspects based on existing text classification models. Wu *et al.* [17] proposed a triplet capsule network with an adjusted triplet loss function to force the network's ability to assign latent features with low discriminative power to the correct class and, therefore, improve classification effectiveness. Xiao *et al.* [20] designed a multi-task learning CapsNet which involves a task routing procedure to cluster shared features by their relevance for different tasks. Liu *et al.* [21] developed a generative explanation framework for text classification that is able to provide human-readable, fine-grained rationale for classification predictions.

V. ANALYSES

This section focuses on the analysis of distinct CapsNet configurations to gain insights about how capsule-based nets inherently work, and how different routing procedures influence model performance for the task of text classification. For that reason, research questions are formulated and subsequently answered based on experimental results.

A. MODEL DEFINITION

The classification model for analyzing diverse CapsNet configurations is illustrated in Fig. 13. The analysis model starts by transforming incoming words into vector representation through the application of a word embedding layer. The subsequent dropout layer ensures that the model learns

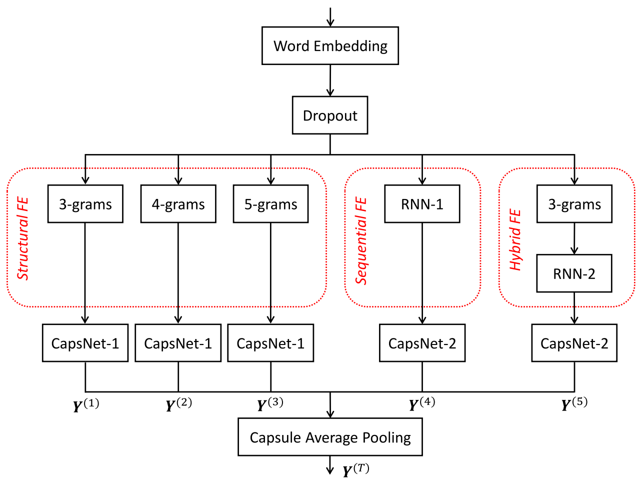

FIGURE 13. Ensemble classification model for CapsNet hyperparameter analysis.

strong representations for all word embedding dimensions. Afterwards, the resulting word embeddings get simultaneously processed by five sub-networks. The entrance of each sub-network corresponds to a representation learning module. The representation learning modules of the first three sub-networks can be categorized as *Structural Feature Extractors (FE)s* since they build n-gram features with distinct granularity levels. In contrast to this, the fourth sub-network extracts sequential features using a preceded RNN. The last sub-network utilizes an *Hybrid FE* which is represented by a mixture of a structural and a sequential FE. Each sub-network ends with its own CapsNet for classification. These CapsNets use identical hyperparameter configurations, defined by the current analysis run. Thus, the analysis model is actually an ensemble approach, which comprises five submodels that can be jointly trained. The total prediction $\mathbf{Y}^{(T)}$ for an incoming sample is finally calculated as the arithmetic mean over all voted sub-predictions $\mathbf{Y}^{(i)}$. Advantages of the designed analysis model are:

- **different feature types** as input for CapsNets
- increasing **confidence in analysis results**
- **computational efficiency** through parallel architecture.

The intention behind the extraction of **different feature types** is to provide a comprehensive consideration about the potential of CapsNets for representing complex entities, independent from the concrete feature kind. Additionally, structural and sequential features are common features in the text domain. The encapsulation of five sub-networks within the basic analysis model significantly increases the **confidence in analysis results**, because the result of each analysis can be interpreted as the averaged result over five *sub-analyses*, through the five sub-networks and the *Capsule Average Pooling (CAP)* layer. This provides a stabilizing effect on the model performance for different runs. Since each sub-network acts as isolated component independent from all other sub-networks, computations of sub-networks can be concurrently applied, which amplifies the **computational efficiency** of the model.

Fig. 14 displays the structure of the used CAP layer. The CAP layer receives as input a list of capsule outputs with equal capsule dimensionality. Then, the CAP layer determines the activation for each class capsule. Here, activations are computed as the length of the instantiation parameter vectors and normalized by the softmax function, applied once per voting. Matrix-shaped capsules are rearranged into vector form before activation calculation, which improves comparability between both capsule models. Finally, the CAP layer returns the arithmetic mean over the activation values for the various classes. The CAP layer does not guarantee a valid probability distributions for the final class prediction, means that the probabilities of all classes does not imperatively sum up to one. Nevertheless, this behavior is not crucial for the subsequently proposed analyses.

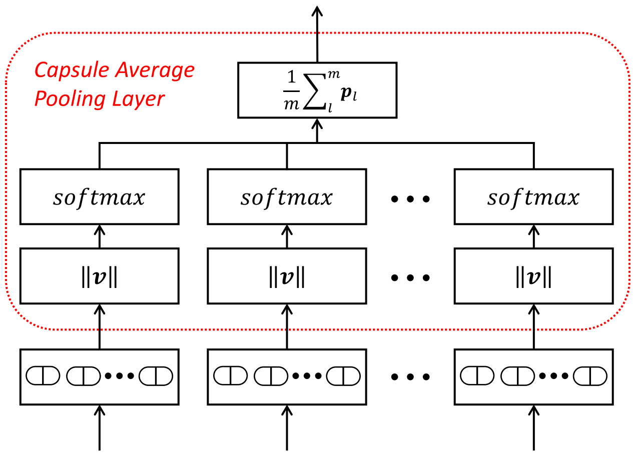

FIGURE 14. Capsule average pooling layer as voting mechanism for predictions from different capsule sub-networks, based on [8], [9].

As a note, a similar parallel CapsNet model for analysis purposes was already proposed by Zhao *et al.* [8]. But Zhao *et al.*'s designed architecture was restricted to the use of structural FEs with different granularity levels. Moreover, the structure of their CapsNet classifiers differs from them used in this paper. (More information about the used classifiers is provided in the remainder of this section.) [8]

Implementing an ensemble of representation learning modules with own classification components, that are jointly trained, should enforce the overall network to learn strong entity representations, as already described by Zhao *et al.* [8]. For that reason, the model accesses the internal structure of input data over three kind of feature extractors: *Structural FE*, *Sequential FE* and *Hybrid FE*.

Subfigure (a) in Fig. 15 visualizes the detailed structure of the components within all feature extractors in the designed classification model: n-Grams are constructed by applying a regular Conv1D with 16 filters, a kernel size of n, a stride of 1 and the Rectified Linear Unit (ReLU) activation function. Both RNN modules consist of Bidirectional

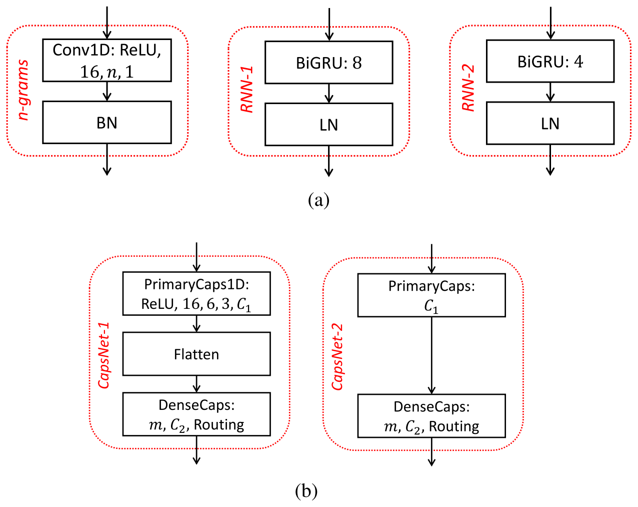

FIGURE 15. (a) Sub-modules for all FEs in the analysis model. (b) Both CapsNet classifiers for the analysis model.

GRUs (BiGRU)s. Since *RNN-2* is combined with a preceding n-gram FE in the analysis model, the number of BiGRUs is halved compared to *RNN-1*. This should help to balance the various FE modules. Although, balancing is difficult because there are three structural FEs in use and only one instance from the other FEs. Furthermore, the hybrid FE has the potential to be much more powerful than the other FEs, since the number of stacked layers significantly increases modeling ability, in general. The hybrid FE in the classification model was inspired by Wang *et al.*'s model [58], which builds a common LSTM layer upon two concatenated convolutional subnetworks. To optimally support the representation learning process in the following analyses, BN is used in combination with CNN layers and LN is applied for RNN layers.

One crucial problem of time-window approaches such as n-grams lies in the choice of the underlying granularity, too small n-grams can ignore relevant contextual information whereas too large n-grams harbor the risk of overfitting on the training data [75]. Because Zhao *et al.*'s analysis [8] supplied meaningful results, the same granularity levels for the n-gram feature extractors are chosen. Thus, 3-grams, 4-grams and 5-grams are utilized.

Suitable for the different FEs in the analysis model, two distinct CapsNet classifiers are defined, as illustrated in subfigure (b). Both CapsNet classifiers are composed of two capsule layers where the prior layer corresponds to PrimaryCaps and the last layer contains the class capsules.

CapsNet-1 represents the classifier for all structural FEs. Since structural FEs provide the input to their classifiers in the form of an one-dimensional input sequence, CapsNet-1 is equipped with an introducing *PrimaryCaps1D* layer. The PrimaryCaps1D layer creates capsules with desired capsule dimensionality by conducting a regular one-dimensional convolution and gathering capsule sequences by concatenating feature maps. This convolutional operation is followed by the application of a BN layer for stabilizing input distributions to the next layer. The PrimaryCaps1D layer in the CapsNet-1 component uses a ReLU nonlinearity, outputs 16 capsule types (correspond to feature maps with capsules), a kernel size of 6 and a stride length of 3. The parameter C_1 means the resulting set of capsules. Afterwards, the capsules C_1 with their current outputs get flattened. Normally, the flatten operation just constitutes a reshaping operation. Finally, the flattened capsule outputs are propagated to the *DenseCaps* layer which embeds the class capsules in a fully-connected feedforward layer. The number of output capsules m is predefined by the used dataset with its specific classification task. Moreover, the DenseCaps layer manages its own *Routing* procedure instance.

The input for the *CapsNet-2* component represents a collection of scalar-outputs produced by a preceding BiGRU layer. Therefore, CapsNet-2 can be connected to both sequential and hybrid FEs. First of all, the CapsNet-2 classifier rearranges the collection of incoming scalar-values to form PrimaryCaps. Then, capsule outputs are sent to a DenseCaps layer with same parameter configuration as the equivalent layer in the CapsNet-1 component.

B. TRAINING SETUP

Table 3 lists the general hyperparameters that are used in each analysis. The hyperparameter list is divided into *Model* and *Training* parameters. Training parameters are only applied during the optimization of a model, whereas model parameters are fixed for a model's lifetime.

TABLE 3. General hyperparameters for all analyses.

Type	Parameter	Value
Model	Mini-batch Size	32
	Embedding Size	16
	Dropout Rate	0.2
	Sequence Length	$\tilde{t}^{(d)}_{avg}$
	Vocabulary Size	$1 + \lvert\mathcal{V}^{(d)}_{train}\rvert$
Training	Epochs	10
	Optimizer	*Adam*
	Loss Function	*Categorical Cross Entropy*
	Early Stopping	Restore best parameters based on validation loss

In general, the mini-batch size of a model is an important hyperparameter which influences training and prediction speed, and resulting classification accuracy. Interestingly, Bhattacharjee [14] experimentally showed for the CBD that varying the mini-batch size had solely a marginal effect on the accuracy of the tested CapsNet. Evidently, this experimental result can only be seen as an indication and not be transferred to all datastets in general. However, one central aim of this paper is to empirically investigate the raw capacity of CapsNets in the text classification domain, without adjusting general hyperparamters to fit best on the six considered datasets. No pretrained word embeddings are utilized for an unbiased analysis and for obtaining task-specific word representations. To prevent overfitting and to enforce the learning of meaningful distributed representations for all considered datasets, a relatively small embedding size of 16 is used. This is also supported by the dropout layer which ignores random 20% of the word embedding dimensions during a training step. The sequence length $\tilde{t}^{(d)}_{avg}$ is the average number of tokens for the regarded dataset d. The average sequence lengths are calculated over the whole datasets for practical simplicity. However, they should be near the same as the average sequence lengths of the associated training sets. The vocabulary size corresponds to the cardinality $|\mathcal{V}^{(d)}_{train}|$ of the vocabulary set within the training data of each dataset d. The (1+) supplement results from one auxiliary token as placeholder for unknown words. Each analysis model with a specific parameter configuration is trained for 10 epochs. The *Adam* [76] optimizer with default parameter configuration in TensorFlow is used for gradient descent optimization. At the moment of this writing, the default learning rate of the Adam optimizer is $\lambda = 0.001$.[13] As loss function the cross entropy is applied which gives a notion about the dissimilarity between a predicted probability distribution and the ground truth [25]. The categorical cross entropy is a special kind of the cross entropy for target labels in the shape of one-hot encodings. Again, the categorical cross entropy is utilized with TensorFlow's default parameter configuration.[14] *Early stopping* represents a simple regularization strategy for preventing overfitting of models by aborting the training process when the error on the validation set does not decrease after a predefined number of epochs [25], [28], [32]. In this work early stopping is only conducted to restore the best learned parameters with respect to the error on the validation set.

All analyses are conducted on a single *64-bit*-machine with an *Intel i7-8550U CPU* with 1.80 *GHz*. The CPU comprises four physical cores and can simultaneously process eight threads. The computer system is equipped with 16 *GB* memory.

The only applied preprocessing to all six datasets includes the tokenization using the already mentioned *TweetTokenizer*[15] from the Python *nltk* module. The tokenization process involves the transformation of all tokens into lowercase. For each analysis, all datasets get completely shuffled and divided into a training, validation and test set with split ratios of 70%, 15% and 15%, respectively.

C. ROUTING PROCEDURE COMPARISON

The goal of this analysis is to provide a thorough overview of the performance of CapsNets with various routing procedures in the text classification domain. All routing procedures which were introduced in the fundamentals of this paper are investigated. In particular, the static routing procedure is included in order to test if routing-by-agreement indeed

[13]https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[14]https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy

[15]https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.casual

has a positive effect on text classification performance. If the use of routing-by-agreement procedures would result in no performance improvement, the excluding of such procedures could save valuable computing resources. Another aspect of this analysis represents the exploration if a routing procedure outperforms the other ones with respect to the six considered datasets. These objectives are summarized in the following research questions (RQ)s:

RQ 1a: *Does the use of routing-by-agreement positively affect text classification performance?*

RQ 1b: *Is there a dominating routing-by-agreement procedure?*

The analysis results for empirically answering the above research questions comprise: the final accuracies for all datasets in Table 4, the mean epoch durations in Table 5 and six plots about the validation loss development in Fig. 16. The specific hyperparameter configuration for this analysis involves the use of vector-shaped capsules with vector lengths of 4 and 8 for the first and second capsule layer, respectively.

TABLE 4. Final percentage accuracies on the corresponding test sets for the routing procedure comparison. The best accuracy achieved for each dataset is emphasized in bold and underlined. All values below 90% are bold and colored in red.

Dataset	Static Routing	Dynamic Routing	Enhanced Dynamic Routing	k-Means Routing	EM Routing
SD	99.67%	99.80%	99.80%	**<u>100.00</u>**%	99.53%
LIAR	**75.86**%	91.92%	91.35%	**<u>92.28</u>**%	**84.57**%
20-NG	**43.14**%	92.68%	**<u>98.37</u>**%	**62.81**%	95.75%
CBD	**<u>99.97</u>**%	99.95%	99.95%	**<u>99.97</u>**%	99.77%
IMDB-RD	96.87%	90.61%	**<u>98.55</u>**%	97.65%	93.64%
AG-NTCD	96.86%	96.01%	**<u>96.87</u>**%	96.80%	96.46%

TABLE 5. Mean durations for one epoch in each training process. All durations are stated in seconds [s].

Dataset	Static Routing	Dynamic Routing	Enhanced Dynamic Routing	k-Means Routing	EM Routing
SD	$4s$	$6s$	$6s$	$5s$	$7s$
LIAR	$7s$	$14s$	$14s$	$11s$	$17s$
20-NG	$401s$	$1017s$	$1054s$	$723s$	$1585s$
CBD	$7s$	$10s$	$11s$	$9s$	$11s$
IMDB-RD	$196s$	$283s$	$287s$	$238s$	$332s$
AG-NTCD	$190s$	$333s$	$348s$	$268s$	$394s$

The mean epoch runtimes in Table 5 are rounded to seconds. The order of the routing procedures from short to long durations is: *Static Routing* → *k-Means Routing* → *Dynamic Routing* → *Enhanced Dynamic Routing* → *EM Routing*. Because all routing-by-agreement procedures use the same number of iterations $i = 3$, the training duration discrepancy completely results from the computational effort of the routing algorithms. These algorithm runtimes also need to be kept in sight when deciding for a routing procedure for a CapsNet model.

Numeric instabilities sometimes occurred when using enhanced dynamic routing or EM routing. Then, it was necessary to partially re-run the corresponding analysis to obtain complete results. In this context, it was particularly difficult to provide numerically stable routing procedure implementations since problems rarely arose, which heavily complicated debugging. This means that, although evidently numeric problems such as *division by zero*, *negative numbers under the root* etc. were prevented, the handling of distributed representations in combination with iterative and computationally intense routing algorithms, though, can lead to numeric instabilities. Further research is desirable for resilient algorithm design of routing-by-agreement procedures.

RQ 1a: The first formulated research question can be answered with: *Yes, but it depends*. According to the analysis results, static routing was inferior to all routing-by-agreement procedures for the datasets LIAR and 20-NG. For all other datasets, the use of static routing led to comparable accuracies to the routing-by-agreement strategies. In classification tasks where static routing achieves sufficient performance, it could be preferred for inter-layer communication, since its computational effort is dramatically lower than for the considered routing-by-agreement algorithms.

One possible reason for the low classification accuracy of static routing on the 20-NG dataset could be that this dataset contains many noise which is not eliminated by an intelligent routing procedure. The 20-NG dataset habors high potential for containing noise, because of its large mean sample length. One argument against this explanation represents the high accuracy reached for the IMDB-RD which has similarly long samples on average. Another reason for the bad performance of static routing on the LIAR and 20-NG datasets could arise from the lack of modeling complex part-to-whole relations. In that sense, the ability to selectively route low-level features to high-level ones could be crucial for solving the text classification tasks of these datasets. Hence, testing distinct routing algorithms within a CapsNet could lead to deep insights about the inherent data structure and the regarded classification task.

In general, the validation loss curves attribute routing-by-agreement procedures a more effective learning process than static routing. In consequence of this, Kim *et al.*'s [10] conjecture that static routing should make a CapsNet for text classification more robust by ignoring complex relationships between low-level and high-level capsules cannot be confirmed based on the analysis results. In summary, the experiments indicate that routing-by-agreement procedures are usually superior to static routing. But if static routing is sufficient for solving a certain task, its use can save valuable computing resources.

RQ 1b: Indeed, enhanced dynamic routing definitely provides dominating results against the rest of the routing algorithms. The classification accuracies of enhanced dynamic routing are constantly high. It reached for the three datasets 20-NG, IMDB-RD and AG-NTCD the best values at all. For the remaining datasets, it also comes close to the best reached accuracies. Moreover, enhanced dynamic routing supplies the best validation loss progresses for all considered datasets, except the LIAR dataset. Its validation loss curve for the LIAR dataset can be categorized as the

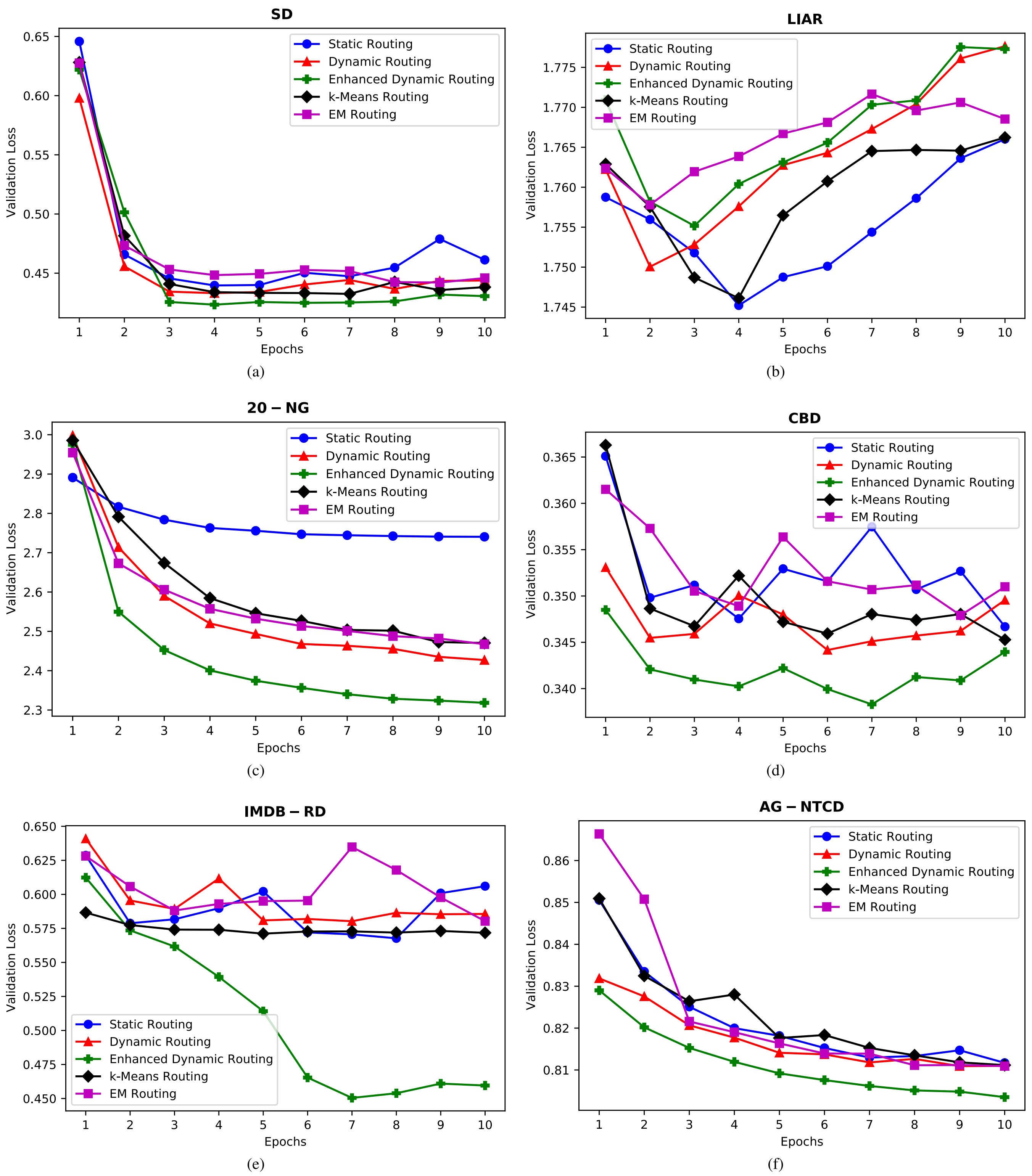

FIGURE 16. Validation loss development for all six datasets over ten epochs. For each dataset the five considered routing procedures are investigated.

second worst because the global loss minimum is the second highest and the model seems to overfit early. Despite these observations, enhanced dynamic routing reaches an high classification accuracy above 90% for the LIAR dataset.

But again, the results emphasize that different routing procedures work well for distinct datasets and tasks. Therefore in a practical application, it is recommended to test various routing procedures in the first instance to gain an intuition

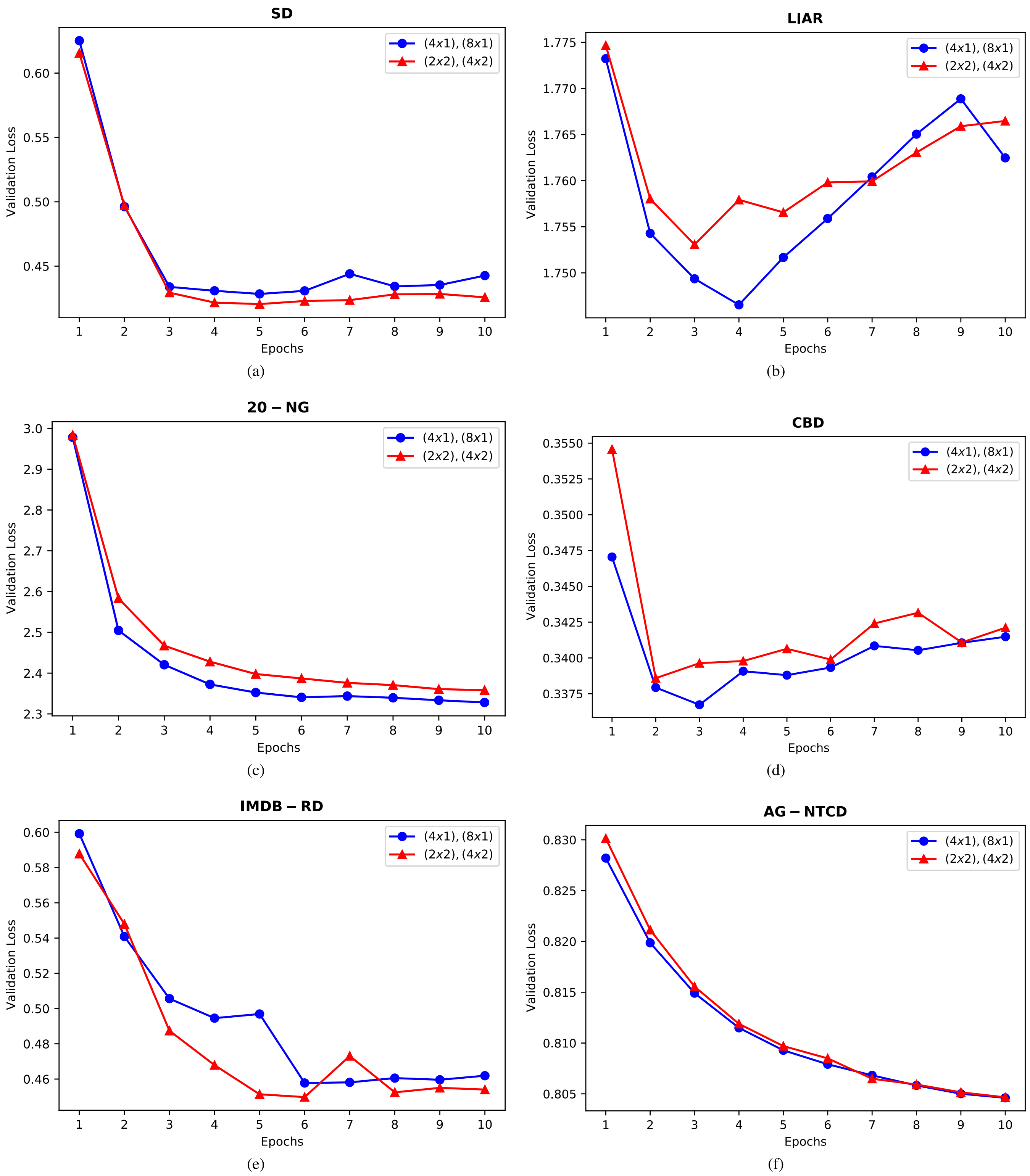

FIGURE 17. Validation loss development for all six datasets over ten epochs. For each dataset the use of vector-shaped versus matrix-shaped capsules is investigated.

about which routing algorithms satisfy the defined application requirements best and should be examined in further tests.

Enhanced dynamic routing will be utilized as routing procedure in all further analyses, since it supplied convincing classification results for all six datasets. Furthermore, it appears to be more robust than static routing, k-Means routing and EM routing because each of these algorithms has a significant deficit in the final accuracies for at least one dataset. In addition, enhanced dynamic routing achieved for each dataset similar or better accuracies compared to the traditional dynamic routing procedure.

D. VECTOR SHAPE VS. MATRIX SHAPE

Matrix-shaped capsules were initially conceived to model 2-D affine transformations to naturally capture 3-D viewpoint variations for image data [2], [4]. The desired affine transformations are inherently applied through multiplication with the transformation matrices which get learned during model training. To the best of our knowledge, this is the first time where matrix-shaped capsules are explored for the text domain. This analysis wants to answer the question if matrix-shaped capsules negatively or positively influence model performance compared to vector-shaped capsules, as proposed in the subsequent research question. For this purpose, two CapsNet configurations get examined. One configuration uses for both capsule layers in the CapsNet classifiers vector-shaped capsules and the second configuration utilizes matrix-shaped capsules.

RQ 2: *Are vector or matrix-shaped capsules preferable for text classification?*

The analysis results for empirically answering the above research question comprise: the final accuracies for all datasets in Table 6 and six plots about the validation loss development in Fig. 17. The first configuration uses vector-shaped capsules with vector length 4 and 8 for both capsule layers, respectively. The second configuration utilizes matrix-shaped capsules with dimensionality (2×2) and (2×4) for the two capsule layers. Thus, both configurations provide for each capsule the equal number of instantiation parameters for the same layers.

Contrary to the works [2], [4], no square matrices are used for all capsule layers. This decision was made because the focus does not lie on enabling 2-D affine transformation operations, which are especially helpful in the image domain, but rather investigating if the kind of *feature composing* and the dimensionality of transformation matrices significantly affects the classification performance of a model.

The number of parameters comprised by transformation matrices varies when vector or matrix-shaped capsules are involved. Equation (10a) and (10b) give an abstract formulation for the multiplication of transformation matrices (colored in red) with the output of vector or matrix-shaped capsules. For instance, the previously defined capsule dimensions for the first CapsNet configuration lead to calculations of the type $(8 \times 4) * (4 \times 1) = (8 \times 1)$. Hence, a transformation matrix with $8 * 4 = 32$ trainable parameters must be created. The second CapsNet configuration encompasses transformation computations of the form $(4 \times 2) * (2 \times 2) = (4 \times 2)$ which results in only $4 * 2 = 8$ trainable parameters per lower to higher-level capsule connection.

TABLE 6. Final percentage accuracies on the corresponding test sets for the comparison between vector and matrix-shaped capsules. The best accuracy achieved for each dataset is emphasized in bold and underlined.

Dataset	(4×1), (8×1)	(2×2), (2×4)
SD	**<u>100.00</u>**%	99.93%
LIAR	**<u>92.18</u>**%	91.50%
20-NG	97.49%	**<u>97.88</u>**%
CBD	**<u>99.95</u>**%	**<u>99.95</u>**%
IMDB-RD	**<u>99.12</u>**%	98.95%
AG-NTCD	96.67%	**<u>96.80</u>**%

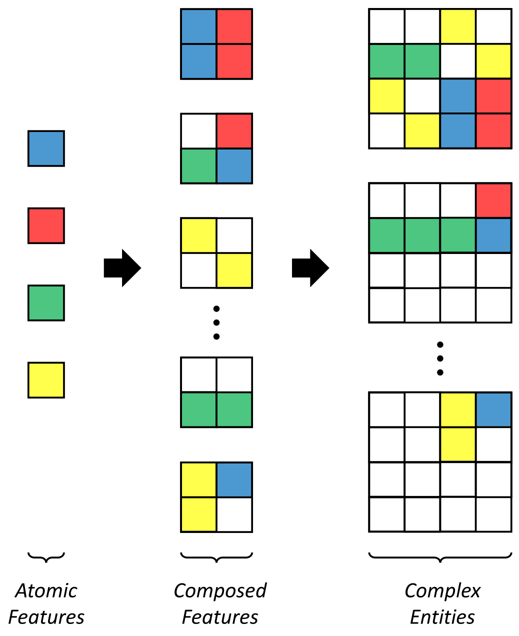

FIGURE 18. Visualization of growth in concept diversity.

Case	Dimensions	
$Vec \rightarrow Vec$	$({\color{red}a_1} \times {\color{red}b_1}) * (b_1 \times 1) = (a_1 \times 1)$	(10a)
$Mat \rightarrow Mat$	$({\color{red}a_2} \times {\color{red}b_2}) * (b_2 \times c_2) = (a_2 \times c_2)$	(10b)

To ensure a kind of comparability between both CapsNet configurations, it was decided to equip each configuration with the same number of instantiation parameters for capsules in each layer. Thus, the vector and matrix-shaped capsules should be able to provide the same expressiveness for entity representation. Therefore, the constraints

$$b_1 \overset{!}{=} b_2 c_2, \quad a_1 \overset{!}{=} a_2 c_2 \tag{11}$$

can be concluded. The number of associated parameters p_i for one transformation matrix within the CapsNet configuration i can be calculated as

$$p_i = a_i b_i. \tag{12}$$

For the first configuration follows: $p_1 = a_1 \, b_1 = a_2 \, b_2 \, c_2^2$. The second CapsNet configuration encapsulates

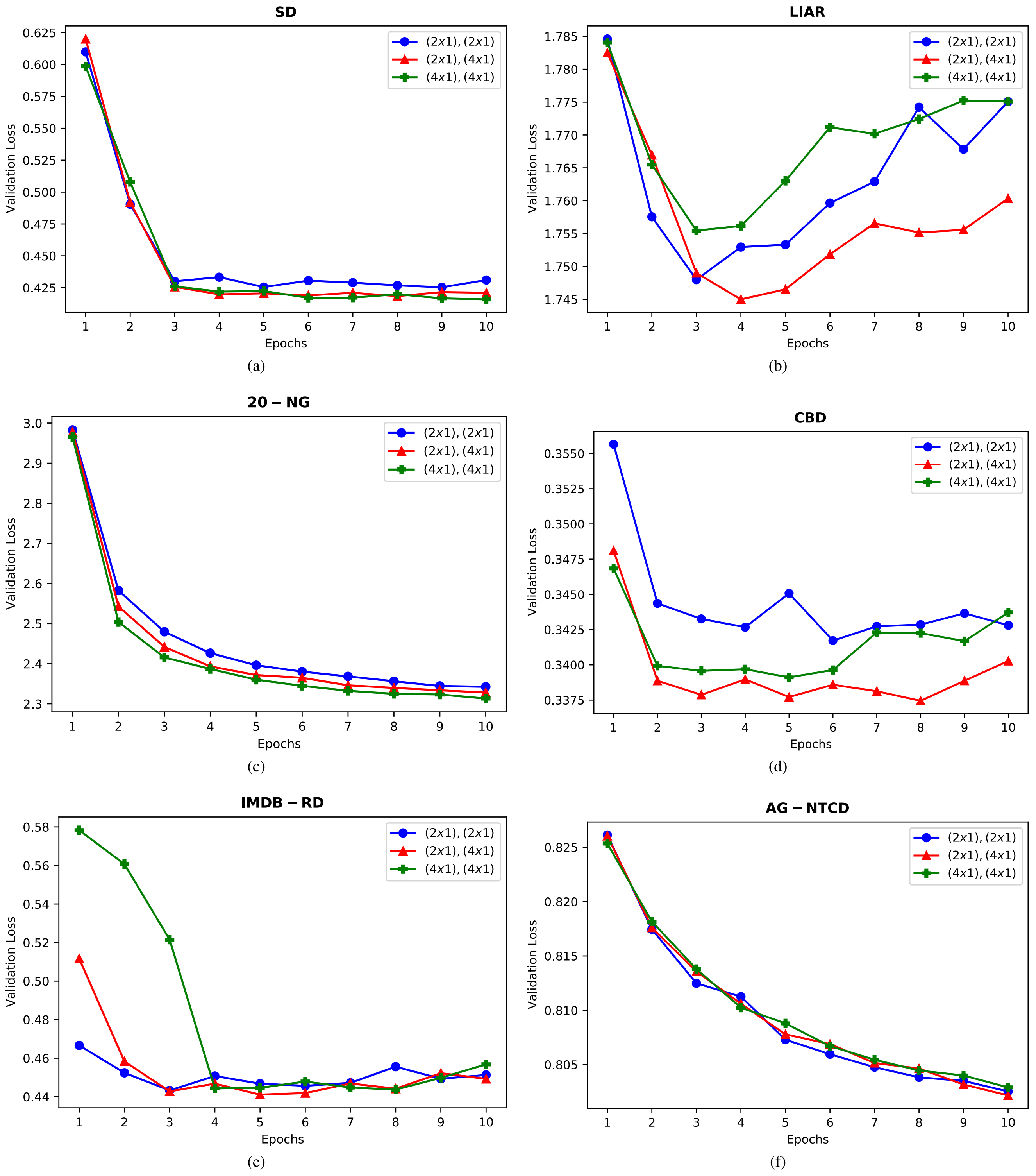

FIGURE 19. Validation loss development for all six datasets over ten epochs. For each dataset the impact of a growing capsule dimensionality within the progress of CapsNets is investigated.

transformation matrices with $p_2 = a_2\, b_2$ trainable parameters. From this abstract definitions for p_1 and p_2 results the relation $p_1 = c_2^2\, p_2$. This means that a two-layered CapsNet with matrix-shaped capsules has $1/c_2^2$ times less parameters per transformation matrix than a CapsNet with vector-shaped capsules and an equivalent number of instantiation parameters.

The term *feature composing* means in this context the difference in linear combinations within a regular matrix multiplication, when vector or matrix-shaped capsules are in use.

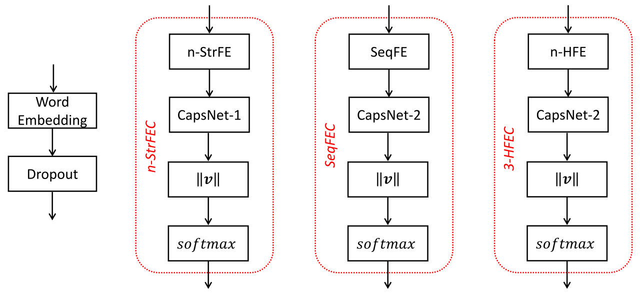

FIGURE 20. Division of the basic analysis model into its FEs with corresponding CapsNet classifiers.

In the case of vector-shaped capsules, each transformation matrix gets multiplied with an instantiation parameter vector. During multiplication, the resulting matrix is compound of the dot products between each row in the transformation matrix and the instantiation parameter vector. In opposite to this, if matrix-shaped capsules are used instead, matrix multiplication is applied by forming the dot product between each row in the transformation matrix and each column in the instantiation parameter matrix. Thus, trainable parameters of the transformation matrix are shared over the columns of an output from a matrix-shaped capsule. This observation agrees with the number of trainable parameters for both CapsNet configurations. Nevertheless, it is difficult to make meaningful statements about the impact of this special way of weight sharing. Intuitively, it seems more contra-productive than helpful because capsules have the ability to store diverse entity properties and using one weight for two or more features sacrifices expressiveness. Applying quantitative analyses also may be difficult since balancing between two possible test configurations would be already tough.

RQ 2: The analysis results show that CapsNets with vector or matrix-shaped capsules both work well for the task of text classification. According to the conducted experiments, the use of vector or matrix-shaped capsules has an marginal impact on the development on validation loss curves during model optimization. Theoretically, the vector-based CapsNet variant has an advantage in model expressiveness over the matrix-based one, through the higher number of learnable parameters in total. However, both CapsNet configurations achieve almost the same final classification accuracy for all examined datasets. Future experiments with larger CapsNet architectures on more complex datasets and classification tasks are desirable for producing more convincing results about the effects of distinct capsule models.

E. GROWING CAPSULE DIMENSIONALITY

Fig. 18 supplies an intuition about how concept diversity may grow within a neural network. This intuition is imparted by illustrating an exemplary step-wise progress in concept development. Fig. 18 is divided into three stages with increasing concept complexity from the left to the right-hand side. At the beginning, some *Atomic Features* are given, visualized as squares with unique colors. If a neural network processes images, atomic features are often pixel intensities. In the case of text classification, atomic features may be single characters or words. The next stage consists of *Composed Features* that are represented as other square units which can contain up to four atomic features. If each position in the four-element grid of a composed feature can be any atomic feature or the absence of a feature then there would exist five options for assignment per grid position. This would lead to $5^4 = 625$ possible composed features in the second stage. The third stage continues this proceeding by forming even more *Complex Entities* after the same pattern as before. So, complex entities again correspond to four-element grids but are now assembled using composed features. Thus the third stage would produce $625^4 \approx 10^{11}$ combinations, which is a tremendous amount of possible concepts that should be captured by a computation model in order to fully *understand* a task-specific domain. Furthermore, this situation already occurs within three stages.

This intuition can be straightforwardly transferred to the development from low-level features up to representing complex entities within neural networks. In that sense, the stages in Fig. 18 would represent three consecutive neural network layers. One could argue that in real use cases neither all possible features/concepts are relevant for the given task nor the feature space is free of noise, which may significantly reduce the whole feature space per neural network layer. Nevertheless, a neural network model usually handles input data with much higher dimensionality than just four atomic features, in the most cases feature spaces are rather real-valued than discrete, features can be duplicated in time or space depending on the input data, and typical neural network architectures are composed of many more layers than three. To name just a few examples. Putting all this together leads to the conjecture that going deeper in a neural network increases the need for representational capacity of features/entities per layer.

This analysis wants to provide evidence about if continuously increasing capsule dimensionality, in the progress of a neural network architecture, has a beneficial impact on classification performance of a CapsNet model. Additionally, this should implicitly prove the assumption that feature/concept diversity grows from layer to layer within a neural network. Sabour *et al.* [3] also recommended to extend the vector dimensionality of capsules with increasing depth of a CapsNet to support the progress in entity complexity. Unfortunately, Sabour *et al.* [3] did not deliver any proof or experiment to confirm their conjecture. For supplying meaningful analysis results, one configuration uses increasing capsule dimensionality, while two further configurations utilize a constant capsule dimensionality for all layers. In particular, the two further configurations serve as baselines to evaluate the effects of the first CapsNet configuration. The corresponding research question is subsequently formulated:

RQ 3: *Should the capsule dimensionality be increased in the progress of a network architecture?*

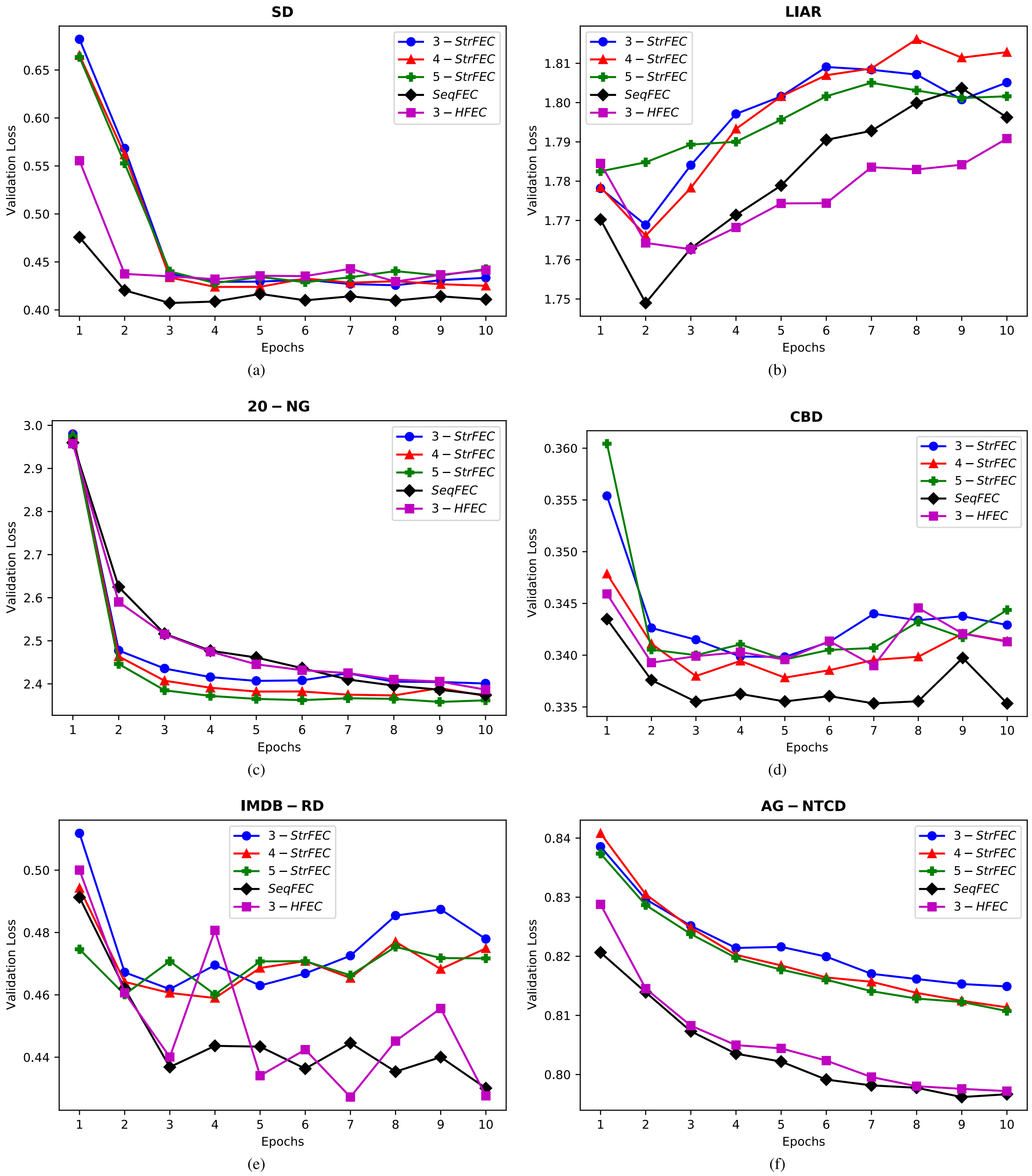

FIGURE 21. **Validation loss development for all six datasets over ten epochs. For each dataset the influence of distinct FEs is investigated.**

The analysis results for empirically answering the above research question comprise: the final accuracies for all datasets in Table 7 and six plots about the validation loss development in Fig. 19. All CapsNets for this analysis are composed of vector-shaped capsules. The first CapsNet configuration with increasing capsule dimensionality uses for its first layer the capsule dimensionality (2×1) and for its second layer (4×1). Therefore, the capsule vector

TABLE 7. Final percentage accuracies on the corresponding test sets for analyzing the impact of a growing capsule dimensionality. The best accuracy achieved for each dataset is emphasized in bold and underlined.

Dataset	(2 × 1), (2 × 1)	(2 × 1), (4 × 1)	(4 × 1), (4 × 1)
SD	<u>**100.00**</u>%	<u>**100.00**</u>%	99.93%
LIAR	90.67%	89.99%	<u>**92.44**</u>%
20-NG	98.51%	96.99%	<u>**98.90**</u>%
CBD	<u>**99.97**</u>%	<u>**99.97**</u>%	99.95%
IMDB-RD	99.16%	<u>**99.17**</u>%	98.56%
AG-NTCD	<u>**97.00**</u>%	96.83%	96.62%

TABLE 8. Final percentage accuracies on the corresponding test sets for analyzing the impact of distinct FEs. The best accuracy achieved for each dataset is emphasized in bold and underlined. All values below 90% are bold and colored in red.

Dataset	3-StrFEC	4-StrFEC	5-StrFEC	SeqFEC	3-HFEC
SD	99.73%	<u>**99.87**</u>%	<u>**99.87**</u>%	99.40%	98.60%
LIAR	90.35%	<u>**90.93**</u>%	<u>**90.93**</u>%	**76.96**%	**59.38**%
20-NG	**82.20**%	90.20%	<u>**93.03**</u>%	**54.64**%	**61.96**%
CBD	99.82%	<u>**99.92**</u>%	99.87%	99.90%	99.59%
IMDB-RD	98.96%	<u>**99.19**</u>%	99.08%	97.32%	96.79%
AG-NTCD	95.20%	95.57%	95.51%	<u>**97.20**</u>%	97.13%

dimensionality is doubled from the first to the second layer. The two CapsNet configurations for the baselines were chosen to give a lower and an upper boundary for the results of the first configuration. The configuration for the lower boundary only contains capsules with dimensionality of (2 × 1). The capsules for both layers within the configuration for the upper boundary have a dimensionality of (4 × 1). Evidently, the CapsNet for providing the lower boundary encompasses less parameters than the first configuration with increasing capsule dimensionality. Analogously, the upper boundary encapsulates more parameters than the first configuration with increasing capsule dimensionality.

RQ 3: This research question can be answered with: *Yes, generally the capsule dimensionality should be increased in the progress of a network architecture*. This answer is grounded on the potential for receiving a positive effect on model optimization when capsule dimensionality grows with the depth of a neural net. Although the analysis results only show beneficial impacts on the validation loss curves for the LIAR dataset and the CBD, it mostly comes for free and can reduce the total number of model parameters. More precisely, the number of parameters can be pruned as side effect because usually neural networks are pyramid-shaped with less units from one layer to the next one. For instance, if a CapsNet has two layers with (4 × 1), (4 × 1) capsule dimensionalities, then using the dimensionalities (2 × 1), (4 × 1) instead can significantly decrease the number of parameters of the whole model and possibly retain model performance. In contrast to this, using the dimensionalities (2×1), (2×1) could meaningfully lower model performance. Of course, this is a theoretical statement which strongly depends on the given dataset and the classification task. However, a growing capsule dimensionality can reduce the number of trainable parameters which has the potential to mitigate the omnipresent problem of overfitting and to improve generalizability of a model. Besides, the conducted analysis does not give indications for negative effects when a growing capsule dimensionality is used. The classification accuracies for the test sets of all datasets in Table 7 show no significant differences. Hence, future analyses with much deeper CapsNets and more demanding classification tasks may be helpful for further investigations about the effects of a growing capsule dimensionality on model performance. Finally, the analysis results can be considered as indicator that feature/concept diversity grows with the depth of a neural network architecture.

F. FEATURE EXTRACTION ABLATION STUDY

This analysis wants to explore performance differences between distinct FEs for the six regarded datasets. For this purpose, each test configuration is reduced to a single FE with CapsNet classifier from the basic analysis model. Fig. 20 illustrates the three FE Classifier (FEC) types: *Structural FEC (StrFEC), Sequential FEC (SeqFEC)* and *Hybrid FEC (HFEC)*. Moreover, each FEC receives its input from an individual word embedding layer followed by a dropout layer. Because of the missing CAP layer which was utilized as final layer in the basic analysis model, it is necessary to additionally calculate class probabilities. This is realized by determining capsule vector lengths and then processing the lengths by a conventional softmax function. This analysis thematizes the below research question:

> RQ 4: *What is the influence of different FEs on the classification performance of CapsNets?*

The analysis results for empirically answering the above research question comprise: the final accuracies for all datasets in Table 8 and six plots about the validation loss development in Fig. 21. All CapsNets for this analysis consist of vector-shaped capsules. For each CapsNet the capsule dimensionalities (2 × 1), (2 × 1) are utilized for both capsule layers, respectively. Each test configuration is compound of one distinct FEC.

RQ 4: The analysis results emphasize that the performance of a CapsNet classifier can strongly vary with the use of different FE types. In general, FEs influence the resulting model performance depending on the given dataset and the regarded classification task. As illustrated in Table 8, structural FEs with coarser n-gram granularity like four or five should be the choice in the first instance, since they supplied stable performance for all six tested datasets. This leads in turn to the conclusion that n-grams are the preferred low-level feature type for CapsNet classifiers in the text domain. For some classification tasks and datasets FEs with sequential components outperform structural ones, as it happened for the AG-NTCD with respect to the final accuracies and validation loss curves. In particular, the SeqFEC caused better validation loss progresses for all datasets, except for the 20-NG dataset. Furthermore, a HFEC has potential for improving model performance in exceptional cases. However, the tested HFEC appears to be often less robust than a totally sequential FE.

TABLE 9. Summary of the central findings for the conducted analyses and possible next steps for future research.

Analysis	Findings	Future Research
Routing Procedure Comparison	• In general, static routing appears to be inferior to routing-by-agreement procedures for the task of text classification. • The use of static routing significantly reduces computational costs compared to routing-by-agreement procedures. • Enhanced dynamic routing should serve as default routing procedure implementation for the task of text classification. • Testing distinct routing procedures in the first instance helps to reason about inherent characteristics of the considered dataset and classification task.	• Resilient algorithm design for routing-by-agreement procedures is desirable.
Vector Shape vs. Matrix Shape	• Both vector-shaped and matrix-shaped capsules seem to be appropriate for the task of text classification.	• Further tests with enlarged CapsNet architectures and more challenging text classification tasks should be conducted for emphasizing possible differences between the impacts of vector-shaped and matrix-shaped capsules on model performance.
Growing Capsule Dimensionality	• In general, the capsule dimensionality should be increased in the progress of a CapsNet architecture for potentially retaining or improving model performance with possibly less parameters. • The analysis results suggest that feature/concept diversity grows with the depth of a neural network.	• Other experiments with deeper CapsNet architectures and more complex text classification tasks should be conducted for further investigating the effects of growing capsule dimensionality.
Feature Extraction Ablation Study	• Coarser n-gram (means $n \in \{4, 5\}$) FEs should serve as default input to CapsNets for the task of text classification. • Performance differences between CapsNet models with structural and sequential FEs appear to have a strong dependence on the concrete text classification task. • Testing distinct feature types in the first instance helps to reason about inherent characteristics of the considered dataset and classification task.	

In practice, both structural and sequential low-level features should be examined in isolation to get a feeling about the importance of distinct feature types. Testing various low-level feature types for a classification task can help to gain knowledge about the structure of latent variables and the problem domain in a general manner.

VI. CONCLUSION

Table 9 recapitulates the central findings of the conducted analyses, mostly formulated as recommendations for the design of CapsNets in the text classification domain, and lists possible next steps for future research. In summary, it can be said that CapsNets combine the powerful methods of distributed entity representations and intelligent routing procedures with the aim to supersede conventional neural networks with single-output neurons and static signal propagation. These modifications can be seen as logical extensions for neural networks to overcome prominent flaws of this technology. For instance, prominent flaws are retaining noise through static signal propagation and the lack of explainabilty caused by non-transparency in routing and *chaotic* representation of entities distributed over entire layers. Moreover, the analysis results of this paper empirically proved that the performance of CapsNets with routing-by-agreement is robust for a broad variety of configurations, datasets and text classification tasks. This significantly reduces the risk of wrong design decisions in the development process of a text classification model. In that sense, CapsNets can be considered as next-generation neural networks, actually. On the other side, it can be argued that CapsNets with routing-by-agreement procedures evidently demand more computational resources than traditional networks. Thus, in text classification tasks where CapsNets with static routing or regular neural networks are sufficient, simpler network approaches should be preferred. However, this paper comprehensively illustrated the potential of CapsNets and routing-by-agreement strategies for the use in the text classification domain. Since CapsNets with routing-by-agreement represent a relatively young technology, future research is desirable for enriching the theory about CapsNets and for further exploring their performance as text classification method.

ACKNOWLEDGMENT

The work is based on excerpts from the master's thesis of Nikolai A. K. Steur.

REFERENCES

[1] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*. [Online]. Available: http://arxiv.org/abs/1603.04467

[2] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 2011, pp. 44–51.
[3] S. Sabour, N. Frosst, and G. E Hinton, "Dynamic routing between capsules," 2017, *arXiv:1710.09829*. [Online]. Available: http://arxiv.org/abs/1710.09829
[4] G. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with EM routing," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–15.
[5] X. Zhang, P. Luo, X. Hu, J. Wang, and J. Zhou, "Research on classification performance of small-scale dataset based on capsule network," in *Proc. 4th Int. Conf. Robot. Artif. Intell. (ICRAI)*, 2018, pp. 24–28.
[6] Y. Feng, X. Zhu, Y. Li, Y. Ruan, and M. Greenspan, "Learning capsule networks with images and text," in *Proc. 32nd Conf. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 1–5.
[7] T. Vijayakumar and R. Vinothkanna, "Capsule network on font style classification," *J. Artif. Intell. Capsule Netw.*, vol. 2, no. 2, pp. 64–76, May 2020.
[8] M. Yang, W. Zhao, J. Ye, Z. Lei, Z. Zhao, and S. Zhang, "Investigating capsule networks with dynamic routing for text classification," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 3110–3119.
[9] S. Srivastava, P. Khurana, and V. Tewari, "Identifying aggression and toxicity in comments using capsule network," in *Proc. 1st Workshop Trolling, Aggression Cyberbullying (TRAC)*, 2018, pp. 98–105.
[10] J. Kim, S. Jang, S. Choi, and E. Park, "Text classification using capsules," 2018, *arXiv:1808.03976*. [Online]. Available: http://arxiv.org/abs/1808.03976
[11] H. Ren and H. Lu, "Compositional coding capsule network with k-means routing for text classification," 2018, *arXiv:1810.09177*. [Online]. Available: http://arxiv.org/abs/1810.09177
[12] W. Zheng, Z. Zheng, H. Wan, and C. Chen, "Dynamically route hierarchical structure representation to attentive capsule for text classification," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 5464–5470.
[13] R. Aly, S. Remus, and C. Biemann, "Hierarchical multi-label classification of text with capsule networks," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics, Student Res. Workshop*, 2019, pp. 323–330.
[14] U. Bhattacharjee, "Capsule network on social media text: An application to automatic detection of clickbaits," in *Proc. 11th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2019, pp. 473–476.
[15] D. K. Jain, R. Jain, Y. Upadhyay, A. Kathuria, and X. Lan, "Deep refinement: Capsule network with attention mechanism-based system for text classification," *Neural Comput. Appl.*, vol. 32, no. 7, pp. 1839–1856, Apr. 2020.
[16] Y. Wang, A. Sun, J. Han, Y. Liu, and X. Zhu, "Sentiment analysis by capsules," in *Proc. World Wide Web Conf. World Wide Web (WWW)*, 2018, pp. 1165–1174.
[17] Y. Wu, J. Li, V. Chen, J. Chang, Z. Ding, and Z. Wang, "Text classification using triplet capsule networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–7.
[18] H. Fei, D. Ji, Y. Zhang, and Y. Ren, "Topic-enhanced capsule network for multi-label emotion classification," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 28, pp. 1839–1848, 2020.
[19] Z. Chen and T. Qian, "Transfer capsule network for aspect level sentiment classification," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 547–556.
[20] L. Xiao, H. Zhang, W. Chen, Y. Wang, and Y. Jin, "MCapsNet: Capsule network for text with multi-task learning," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 4565–4574.
[21] H. Liu, Q. Yin, and W. Y. Wang, "Towards explainable NLP: A generative explanation framework for text classification," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 5570–5581.
[22] D. Q. Nguyen, T. Vu, T. D. Nguyen, D. Q. Nguyen, and D. Phung, "A capsule network-based embedding model for knowledge graph completion and search personalization," 2018, *arXiv:1808.04122*. [Online]. Available: http://arxiv.org/abs/1808.04122
[23] Y. Wang, L. Huang, S. Jiang, Y. Wang, J. Zou, H. Fu, and S. Yang, "Capsule networks showed excellent performance in the classification of hERG blockers/nonblockers," *Frontiers Pharmacol.*, vol. 10, p. 1631, Jan. 2020.
[24] N. J. Nilsson, "Introduction to machine learning: An early draft of a proposed textbook," in *Draft of Notes for a Standford course on Machine Learning*, 2005, pp. 9–11. Accessed: Oct. 26, 2020. [Online]. Available: https://ai.stanford.edu/~nilsson/mlbook.html
[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, pp. 21–31, 70–73, 239–242, 363–372, 397–401, and 448–543.
[26] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the *EM* algorithm," *J. Roy. Stat. Soc., Ser. B, Methodol.*, vol. 39, no. 1, pp. 1–22, 1977.
[27] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed. Amsterdam, The Netherlands: Elsevier, 2009, pp. 45–46, 741–745, and 807–808.
[28] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006, pp. 259–261, 423–444, and 605–607.
[29] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[30] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, vol. 1, 1967, pp. 281–297.
[31] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.
[32] A. Géron, *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. Newton, MA, USA: O'Reilly Media, 2019, pp. 141–142 and 279–607.
[33] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. London, U.K.: Pearson, 2016, pp. 860–885 and 888–890.
[34] R. Davidson and J. G. Mackinnon, *Econometric Theory and Methods*. London, U.K.: Oxford Univ. Press, 2009, pp. 270–271.
[35] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2018, pp. 41–76.
[36] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2009, pp. 27–28 and 117–119. [Online]. Available: http://www.informationretrieval.org/
[37] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, *arXiv:1511.07122*. [Online]. Available: http://arxiv.org/abs/1511.07122
[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.
[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[40] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, 2015, pp. 448–456.
[41] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," in *Proc. Deep Learning Symp. (NIPS)*, 2016, pp. 1–14. [Online]. Available: arXiv: 1607.06450v1
[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[43] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.
[44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15. [Online]. Available: arXiv: 1409.0473v7
[45] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1412–1421.
[46] T. Mikolov, W.-T. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2013, pp. 746–751.
[47] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[48] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2013, pp. 1–12. [Online]. Available: arXiv: 1301.3781v3
[49] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.
[50] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 2, 2013, pp. 3111–3119.
[51] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *J. Document.*, vol. 28, no. 1, pp. 11–21, 1972.

[52] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, ''Bag of tricks for efficient text classification,'' in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics*, vol. 2, 2017, pp. 427–431.
[53] A. Gkanogiannis and T. Kalamboukis, ''A perceptron-like linear supervised algorithm for text classification,'' in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2010, pp. 86–97.
[54] A. McCallum and K. Nigam, ''A comparison of event models for naive Bayes text classification,'' in *Proc. AAAI Workshop, Learn. Text Categorization*, 1998, pp. 41–48.
[55] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, ''LIBLINEAR: A library for large linear classification,'' *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, Jun. 2008.
[56] T. Joachims, ''Text categorization with support vector machines: Learning with many relevant features,'' in *Proc. Eur. Conf. Mach. Learn. (ECML)*, 1998, pp. 137–142.
[57] H. Hu, M. Liao, C. Zhang, and Y. Jing, ''Text classification based recurrent neural network,'' in *Proc. IEEE 5th Inf. Technol. Mechatronics Eng. Conf. (ITOEC)*, Jun. 2020, pp. 652–655.
[58] R. Wang, Z. Li, J. Cao, T. Chen, and L. Wang, ''Convolutional recurrent neural networks for text classification,'' in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–6.
[59] P. Liu, X. Qiu, and X. Huang, ''Recurrent neural network for text classification with multi-task learning,'' in *Proc. 25th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2016, pp. 2873–2879.
[60] Y. Kim, ''Convolutional neural networks for sentence classification,'' in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1746–1751.
[61] X. Zhang, J. Zhao, and Y. LeCun, ''Character-level convolutional networks for text classification,'' in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 28, 2015, pp. 649–657.
[62] A. Conneau, H. Schwenk, L. Barrault, and Y. LeCun, ''Very deep convolutional networks for text classification,'' in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics (EACL)*, Valencia, Spain, vol. 1, Apr. 2017, pp. 1107–1116.
[63] A. M. Dai and Q. V. Le, ''Semi-supervised sequence learning,'' in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 3079–3087. [Online]. Available: arXiv: 1511.01432v1.
[64] M. Schuster and K. K. Paliwal, ''Bidirectional recurrent neural networks,'' *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[65] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, ''Hierarchical attention networks for document classification,'' in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2016, pp. 1480–1489.
[66] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, ''Attention is all you need,'' in *Proc. 31st Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5998–6008.
[67] B. Pang and L. Lee, ''A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts,'' in *Proc. 42nd Annu. Meeting Assoc. Comput. Linguistics (ACL)*, 2004, p. 271.
[68] W. Y. Wang, '''Liar, liar pants on fire': A new benchmark dataset for fake news detection,'' in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics Short Papers*, vol. 2, 2017, pp. 422–426.
[69] K. Lang, ''Newsweeder: Learning to filter netnews,'' in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 331–339.
[70] A. Chakraborty, B. Paranjape, S. Kakarla, and N. Ganguly, ''Stop clickbait: Detecting and preventing clickbaits in online news media,'' in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Aug. 2016, pp. 9–16.
[71] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, ''Learning word vectors for sentiment analysis,'' *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics*, 2011, pp. 142–150.
[72] *TensorFlow Datasets, a Collection of Ready-to-Use Datasets*. Accessed: Jan. 16, 2021. [Online]. Available: https://www.tensorflow.org/datasets
[73] G. M. Del Corso, A. Gullí, and F. Romani, ''Ranking a stream of news,'' in *Proc. 14th Int. Conf. World Wide Web (WWW)*, 2005, pp. 97–106.
[74] A. Gulli, ''The anatomy of a news search engine,'' in *Proc. Special Interest Tracks Posters 14th Int. Conf. World Wide Web (WWW)*, 2005, pp. 880–881.
[75] A. Graves and J. Schmidhuber, ''Framewise phoneme classification with bidirectional LSTM and other neural network architectures,'' *Neural Netw.*, vol. 18, no. 5, pp. 602–610, 2005.
[76] D. P. Kingma and J. L. Ba, ''Adam: A method for stochastic optimization,'' in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–15. [Online]. Available: arXiv: 1412.6980v9.

NIKOLAI A. K. STEUR received the B.Sc. degree in computer science from Baden-Wuerttemberg Cooperative State University, Mosbach, Germany, in 2018, and the M.Sc. degree with distinction in computer science from Ulm University, Ulm, Germany, in 2021.

His research interests include algorithms and methods for data science, machine learning, artificial intelligence, and the state-of-art development of artificial neural networks in various application domains. He was awarded for the Best Presentation from the Conference Committee at ICACTE, in 2018.

FRIEDHELM SCHWENKER (Member, IEEE) received the Diploma and Ph.D. degrees from the University of Osnabrück.

He is currently a Professor in computer science with the Institute of Neural Information Processing, Ulm University. He has (co-)edited over 20 special issues and workshop proceedings published in international journals and publishing companies, and published more than 250 papers at international conferences and journals. His research interests include artificial neural networks, machine learning, statistical learning theory, data mining, pattern recognition, information fusion, and affective computing.

Dr. Schwenker served as the (Co-)Chair for the IAPR TC3 on Neural Networks and Computational Intelligence. He was the chair from 2016 to 2020. He founded the IAPR TC9 on Pattern Recognition in Human–Computer Interaction.