Ladder Capsule Network

Taewon Jeong 1  Youngmin Lee 1  Heeyoung Kim 1

Abstract

We propose a new architecture of the capsule network called the ladder capsule network, which has an alternative building block to the dynamic routing algorithm in the capsule network (Sabour et al., 2017). Motivated by the need for using only important capsules during training for robust performance, we first introduce a new layer called the pruning layer, which removes irrelevant capsules. Based on the selected capsules, we construct higher-level capsule outputs. Subsequently, to capture the part-whole spatial relationships, we introduce another new layer called the ladder layer, the outputs of which are regressed lower-level capsule outputs from higher-level capsules. Unlike the capsule network adopting the routing-by-agreement, the ladder capsule network uses backpropagation from a loss function to reconstruct the lower-level capsule outputs from higher-level capsules; thus, the ladder layer implements the reverse directional inference of the agreement/disagreement mechanism of the capsule network. The experiments on MNIST demonstrate that the ladder capsule network learns an equivariant representation and improves the capability to extrapolate or generalize to pose variations.

1. Introduction

The convolutional neural network (CNN) has shown superhuman performances over the recent years for a wide range of computer vision tasks such as image classification, segmentation, detection, and tracking. In essence, a CNN predicts whether an object (e.g., face) exists by detecting the existence of features or object parts (e.g., eye, nose, mouth). However, CNN only captures the existence of features, and fails to capture the intrinsic spatial relationship between a part and a whole (e.g., the correct positions of the eye, nose, and mouth to form a face). This problem in CNN is due to the max-pooling that discards the information about the pose (position, size, orientation) of features, although it contributes to the extraction of translation-invariant features. As an alternative to the CNN, the capsule network (CapsNet), a new network architecture recently introduced by Sabour et al. (2017), learns an equivariant representation that is more robust to pose variations. CapsNet captures various pose information of the same feature by replacing scalar-output neurons in the CNN with vector-output capsules, and captures part-whole spatial relationships by replacing the max-pooling in the CNN with a dynamic routing algorithm. CapsNets have shown to outperform CNNs on digit recognition, even when using a dataset of highly overlapping digits (Sabour et al., 2017).

Although the dynamic routing algorithm has shown that it is effective in capturing part-whole relationships, it is inevitable that information on unnecessary lower-layer capsules would be included when constructing higher-layer capsules owing to the nature of the algorithm that expresses the higher-layer capsules by the weighted sum of many lower-layer capsules. Too many unnecessary capsules, similar to other over-parameterized deep learning networks, can cause confusion in delivering the necessary information to the upper layers and can ultimately lead to difficulties in performing the desired tasks (Costa et al., 2002).

In this paper, we propose a new architecture of capsule networks referred to as the ladder capsule network (L-CapsNet) based on an alternative building structure to the dynamic routing algorithm. We first demonstrate via experiments that not all lower-layer capsules are necessarily required to construct higher-layer capsules; only part of the capsules is sufficient to construct higher-layer capsules. To address this finding in the L-CapsNet, we introduce a new layer called the pruning layer, which is inspired by pruning methods that primarily aim at reducing the size of deep neural networks. The pruning layer removes the capsules with small activities and only uses the capsules with large activities. Subsequently, we can construct the higher-level capsules as a linear combination of the selected capsules from the pruning layer, similar to the CapsNet. Unlike the CapsNet that uses an iterative routing based on the agree-
Algorithm 1 Dynamic routing algorithm (Sabour et al., 2017)

Initialize logit parameters $b_{ij} = 0$ for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l + 1)$.

1: for $1:MaxIter$ do
2: \[ c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{kj})} \] for all capsule $i$ in layer $l$.
3: \[ s_j = \sum_i c_{ij} \hat{u}_{j|i} \] and \[ v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} s_j \] for all capsule $j$ in layer $(l + 1)$.
4: \[ b_{ij} = b_{ij} + \langle \hat{u}_{j|i}, v_j \rangle \] for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l + 1)$.
5: end for

In the dynamic routing algorithm, two points are to be emphasized. One is the linear relationship between the lower-level and higher-level capsules. The prediction vector $\hat{u}_{j|i}$ means the prediction of pose of entity $j$ based on the pose of entity $i$. For example, if we know where someone’s nose is, we can predict where his/her face is. Furthermore, if his/her nose is moved to some direction, the predicted position of his/her face is also moved to the same direction. Hence, it is reasonable to assume a linear relationship between the lower-level and higher-level capsules. This assumption is also used in the L-CapsNet to construct higher-level capsules based on lower-level capsules.

The other is the core idea of the dynamic routing algorithm. When we consider the pose of entity $j$, the core parts of entity $j$ in the layer below should predict the pose of entity $j$ consistently, but irrelevant parts are likely to predict the pose differently. Using the example above again, the nose and eyes are core parts of the face; therefore, the predictions of the face pose based on the poses of the nose and eyes should be similar: their predictions should agree. In contrast, the predictions of the face pose from the poses of a pencil or laptop (i.e., irrelevant parts) should disagree. Using this idea, the dynamic routing algorithm is expected to capture the part-whole relationship (lower-level and higher-level capsule relationship). In the algorithm, $c_{ij}$ measures the importance of entity $i$ in constructing entity $j$ in the layer above. Moreover, a high value of $c_{ij}$ indicates that the pose of entity $j$ is similar with the predicted pose from entity $i$.

2.2. Pruning Techniques

Pruning is a method used to reduce network complexity by removing certain unimportant weights, neurons, or channels on the network (Luo et al., 2017). The primary purpose of pruning is to reduce the size of the network such that it can be used for devices with limited computing or storage capacity. For example, Molchanov et al. (2016) developed a greedy criteria-based pruning method that uses the Taylor expansion that approximates the parameter-importance evaluation. Aghasi et al. (2017) and Dong et al. (2017) proposed a layer-wise pruning method to reduce the complexity of deep neural networks. Structured pruning having various scale levels such as feature maps or kernels was used by Anwar et al. (2017) for the real-time application of a deep learning model. Pruning enabled the studies above to show similar or even better performance using a much smaller scale network than a larger network, as it could improve the generalization ability of deep learning models (Thodberg, 1991; Reed, 1993; Augusta & Kathirvalavakumar, 2013). The overfitting caused by a large number of parameters could be prevented by removing unnecessary connections in the networks.

For a similar objective, to improve the generalization of the capsule network, we use pruning as a layer that removes the information of unimportant capsules and retains only capsules that contain important information. This approach...
is similar to the stochastic activation pruning proposed for advanced defense (Dhillon et al., 2018), in that the pruning is based on the activity levels of neurons in the forward direction of the network. The operation of the pruning layer is similar to the $k$-max pooling (Kalchbrenner et al., 2014), which is a generalization of max pooling. However, one major difference exists: our pruning layer does not lose spatial information because the location information of the capsules is transferred to the upper layer by the so-called code vector. More detailed descriptions of the pruning layer and the code vector are presented in Section 3.1.

2.3. Ladder Networks

Over the recent years, many studies have demonstrated that supervised learning with auxiliary unsupervised representation learning can improve the network performance for supervised tasks (Suddarth & Kergosien, 1990). A ladder network, which adds an auxiliary task in the intermediate representation, is one example that is widely used in supervised (or semi-supervised) learning and showed remarkable performance on several tasks (Pezeshki et al., 2016; Dosovitskiy & Brox, 2016). Most ladder networks add auxiliary decoding layers, each of which corresponds to an encoding layer in the network for supervised learning, and each decoding layer targets at reconstructing the output of the corresponding encoding layer. Previous studies showed that this type of networks not only improves representation learning (Sønderby et al., 2016), but also achieves enhanced performance for supervised tasks (Zhang et al., 2016). The CapsNet (Sabour et al., 2017) also uses an additional reconstruction loss to encourage the digit capsules to encode the instantiation parameters of the input digit, which results in the improved performance of digit recognition, compared with the results from using only the margin loss.

Inspired by these, we adopt a ladder structure into the capsule network by introducing a new layer called the ladder layer, which forms an alternative building block to the dynamic routing algorithm. Unlike previous studies, our ladder directly links to supervised tasks and facilitates in capturing the part-whole relationship between capsules. This architecture is similar to the "what-where" autoencoder (Zhao et al., 2015), for which the "where" component is similar to the code vector that is introduced in the L-CapsNet. We discuss the ladder layer in more detail in Section 3.3.

3. Components of L-CapsNet

In this section, we describe the components of the L-CapsNet. The L-CapsNet consists of three components: pruning layer, weight construction and propagation layer, and ladder layer.

3.1. Pruning Layer

The use of pruning in the capsule network was motivated by a simple thought experiment. Let us use an example of a classification task between a human face and a car. The core entities of the face in the lower level could be the nose, eyes, or mouth, whereas the car’s core entities could be wheels, roof, or mirror. If an input image corresponds to a human face, the routing algorithm is expected to predict the face pose based on the low-level capsules of the nose, eyes, and mouth. However, the predictions based on the wheel, roof, and mirror are also executed in the dynamic routing, even though their capsules are not activated. In fact, we only need to consider the results of the agreement/disagreement between the core entities of the face; predictions based on the car’s capsules are not required.

This argument can be shown from our simple experiment to implement the CapsNet with the same architecture as in Sabour et al. (2017) on the MNIST dataset. After training the network, we found that the activities of some capsules were significantly higher than those of others. In addition, these highly activated capsules ($u_i$) tended to have relatively large coupling coefficients ($c_{ij}$) for the desired parent $v_j$. This indicates that the construction of higher-level capsules is primarily contributed by highly activated lower-level capsules; hence, the capsules with low activities need not be emphasized. Figure 1 illustrates the results with the digit “0”. The graph shows the average length of $u_i$ and the average value of $c_{ij}$ of the 100 most active capsules over 5444 samples (total number of the digit “0” in the training set) on the left-$y$-axis by the blue bar and the right-$y$ axis by the red bar, respectively.

![Figure 1: Length of $u_i$ (left $y$-axis, blue bar) and value of $c_{ij}$ (right $y$-axis, red bar) of the 100 most active lower-level capsules to predict the “0” digit capsule.](image-url)

Inspired by these results, we introduce the pruning layer, which implements the selection of important lower-level capsules to ensure that only the outputs of the important capsules are sent to the layer above. We expect that the pruning layer not only reduces computational burden, but also
improves the network generalization. The pruning layer collects the outputs of the $K$ most active capsules. More specifically, consider the outputs of the level $l$ capsules, $U^l = \{u_i \in B^d \mid i = 1, 2, \ldots, n_l\}$, where $B^d$ is the unit ball in $R^d$, and the corresponding activity level (or the probability of existence of an entity), $A^l = \{0 \leq a_i \leq 1 \mid i = 1, 2, \ldots, n_l\}$. The orientation of $u_i$ represents the pose of an entity, and the length of $u_i$ represents the activity level, or the probability of existence of the entity, i.e., $a_i = \|u_i\|$. Before propagating the outputs in level $l$ to level $(l + 1)$, we select the $K$ most active capsules. Let $a(i)$ denote the $i$th highest activity level in $A^l$ such that $a(1) > a(2) > \ldots > a(n)$. Further, let $I(m)$ denote the ordering index for the $m$th lowest activity level within $\{i \mid a_i \geq a(K)\}$. Subsequently, we can collect the outputs of the $K$ most active capsules as $\{u_{I(m)} \in R^d \mid m = 1, 2, \ldots, K\}$. Based on this collection, we can construct $U^l_K \in R^{K \times d}$ matrix, where the $p$th row is equal to $u_{I(p)}$. We also consider one-hot encoded vectors $c^l \in \{0, 1\}^{n_l}$ from the index set $\{i \mid a_i \geq a(K)\}$, which we call the code vector. The code vector restores the information about which capsules are selected, and it contains the information about which capsules in $U^l$ are used to form $U^l_K$. For level $(l + 1)$, we only propagate $U^l_K$ and $c^l$, instead of the whole capsule output $U^l$; subsequently, we construct higher-level capsules in level $(l + 1)$. Figure 2 illustrates an example of the pruning layer with $n_l = 6$ and $K = 3$; when the three capsules $u_1, u_4,$ and $u_6$ are selected among $u_i, i = 1, \ldots, 6$, the code vector becomes $c^l = (1, 0, 0, 1, 0, 1)$.

### 3.2. Weight Construction and Propagation Layer

We propagate each row of $U^l_K$ to higher-level capsules by applying a linear operator, as discussed in Section 2.1. Recall that the CapsNet sets the prediction matrix $W_{ij}$ as a linear operator between all lower-level and higher-level capsules. That is, predictions $\hat{u}_{j|i}$ from all lower-level capsules by multiplying $W_{ij}$ are propagated to the layer above to construct higher-level capsule outputs. In contrast, the L-CapsNet propagates only part of the capsules selected in the pruning layer, $U^l_K$, to the next level above; thus, the same operation with $W_{ij}$ cannot be directly used. Instead of $W_{ij}$, we use the code vector $c^l$, which contains the information about which capsules are highly active. We define a function of the code vector $\{f_{ji}(c^l) \in R^{d \times D}\}$, which is a linear operator for propagating the $i$th row in $U^l_K$ to the capsule $j$ in layer $(l + 1)$.

Moreover, we define another non-negative function of the code vector $p_j(c^l) = (p_{j|1}(c^l), p_{j|2}(c^l), \ldots, p_{j|K}(c^l))$, which determines how much contribution comes from different output vectors of the layer below. Let $u^l_{ij}$ denote the $i$th row of $U^l_K$ and $\hat{u}_{ji}$ denote the prediction made by a capsule $i$. We have $\hat{u}_{ji} = u^l_{ij}f_{ji} \in R^D$. Subsequently, we compute a partial input of capsule $j$ in layer $(l + 1)$, denoted by $s^{l+1}_j$, as follows:

$$\hat{s}^{l+1}_j = \sum_{i=1}^K p_{j|i}(c^l)\hat{u}_{ji} \in R^D. \tag{1}$$

We used the word “partial,” because other components (that will be discussed in Section 3.3) are also used to construct the output of capsule $j$ in layer $l + 1$, $u^{l+1}_j$. More precisely, $s^{l+1}_j$ only represents the pose of entity $j$ in layer $l + 1$. In the layer above, we obtain the activity level and by multiplying it by $\hat{s}^{l+1}_j$, we compute the total output of $v^{l+1}_j$. In the L-CapsNet, the functions of the code vector, $f_{ji}$ and $p_j$, are architecture from a deep convolutional neural network; we train all parameters of the network via backpropagation. This is different from the dynamic routing algorithm: we train the contribution rate of propagation, $p_j(c^l)$, while the dynamic routing determines $c_{ij}$ using an iterative routing algorithm.

### 3.3. Ladder Layer

The ladder layer is designed to capture the part-whole relationship; it plays a similar role as the dynamic routing algorithm, but is based on a different idea. Recall that in the dynamic routing algorithm, predictions made from lower-level capsules are investigated if they agree. The L-CapsNet focuses on the reverse directional inference; lower-level capsules are regressed from higher-level capsules. The primary idea of the ladder layer is that if higher-level capsules are well constructed, we can subsequently infer the pose of the core entities in the layer below. Using the simple example of Section 2.1, if we know the face pose such as the location and orientation, subsequently we can well infer the pose of the core entities of the face, nose, and eyes; however, it is difficult to infer the pose of irrelevant entities such as a pencil or laptop. Recall that $U^l_K$ is constructed by selecting the $K$ most active capsules, which correspond to the entities with high probability of existence. If the elements of $U^l_K$ are indeed the core entities of capsule $j$ in layer $l + 1$, then we can expect that regression from $s^{l+1}_j$ to $U^l_K$ performs well.

As the lower-level and higher-level capsules are linked by a linear relationship, the regression from $s^{l+1}_j$ to $U^l_K$ is also assumed to be linear. We introduce a linear regression operator $f_j^{reg}(c^l) \in R^{K \times d \times D}$, which is another function of the code vector used for the regression of $U^l_K$ from $v^{l+1}_j$. The regressed $U^l_K$ from $v^{l+1}_j$ is denoted by $\hat{u}^{reg}_j$.

$$\hat{u}^{reg}_j = f_j^{reg}(c^l)(s^{l+1}_j)^T \in R^{K \times d}. \tag{2}$$

As $u^l_j$ lies on the unit ball $B^d$, we apply the squashing function to $\hat{u}^{reg}_j$ for down scaling, resulting in $u^{reg}_j$ in Eq.(3). Although the squashing function is nonlinear, it preserves
we choose the hyperparameter

\[ u_j^{reg} = \frac{||u_j^{reg}||^2}{1 + ||u_j^{reg}||^2} u_j^{reg} \in B^{K \times d}. \]  

(3)

Subsequently, we define an L2-norm-based similarity measure, denoted by \( d_j \), between \( u_j^{reg} \) and \( U_K^l \). The measure \( d_j \) allows us to numerically evaluate the part-whole relationship between lower-level and higher-level capsules:

\[ d_j = \exp(-\gamma ||u_j^{reg} - U_K^l||^2), \]  

(4)

where \( ||u_j^{reg} - U_K^l||^2 = \frac{1}{K} \sum_{i=1}^{K} ||u_{ij}^{reg} - U_i^l||^2 \), where \( u_{ij}^{reg} \) denotes the \( i \)th row of \( u_j^{reg} \), and \( \gamma \) is a non-negative hyperparameter. A value of \( d_j \) close to one indicates a good fit of the regression model, resulting in lower-level entities appropriately posed for constructing higher-level entities. Hinton et al. (2018) indicated the importance of sensitiveness to the difference between good and very good agreement, which is one of deficiencies of the dynamic routing algorithm. Hence, we choose the hyperparameter \( \gamma = 5.12 \), which solves for \( \exp(-0.01\gamma) = 0.95 \) to ensure a good fit of the regression model for a well-suited relationship between lower-level and higher-level entities. We let \( d_j \) represent the activity of the capsule \( j \) in layer \( (l + 1) \). Using \( d_j \), the total output of capsule \( j \) in layer \( l + 1 \) is given by \( u_j^{l+1} = d_j \frac{s_j^{l+1}}{||s_j^{l+1}||} \).

Recall that \( s_j \) provides the information about which entity’s pose is represented in each row of \( U_K^l \). Therefore, \( s_j \) informs \( f_j^{reg} \) about the entities that should be regressed from \( s_j^{l+1} \), which makes the \( i \)th row of \( u_j^{reg} \) reconstruct \( u_i^l \). It is noteworthy that the ladder layer trains all weights via backpropagation, unlike the CapsNet that uses an iterative routing. This reduces the computational cost of the L-CapsNet. A comparison of computation time between the CapsNet and L-CapsNet is presented in Section 5.2.

4. Loss on L-CapsNet

Sabour et al. (2017) proposed the margin loss for classification using the capsule network. We also applied this loss, which is expressed on the L-CapsNet as follows:

\[ L_m^{margin} = T_k \max(0, m^+ - d_k)^2 \]  

\[ + \lambda(1 - T_k) \max(0, d_k - m^-)^2, \]

where \( T_k \)'s are outputs of the one-hot encoded labels; \( m^+ \), \( m^- \), and \( \lambda \) are non-negative hyperparameters; \( d_i \) is the activity level of capsule \( i \) in the final layer; further, the number of capsules in the final layer should be the same as the number of labels. In Section 5, we trained the L-CapsNet with the margin loss with \( m^+ = 0.9, m^- = 0.1 \), and \( \lambda = 0.5 \). In addition, we found that adding the loss of difference between the code vector and lower-level activity level, \( ||c^l - A^l||^2 \), would be helpful for training; thus we trained the L-CapsNet with the loss

\[ L = L_m^{margin} + \epsilon ||c^l - A^l||^2 \]

with \( \epsilon = 0.0001 \). We used the Adam optimizer with exponentially decaying learning rate starting from 0.001.

5. L-CapsNet Architecture and Experiments

5.1. L-CapsNet Architecture

The general architecture of the L-CapsNet is depicted in Figure 3. We start with a convolutional layer having a 9 x 9 kernel and 256 channels with a stride of 1 and the ReLu activation function. This layer propagates the activities of the local feature detectors to the primary capsule layer, similar to the work of Sabour et al. (2017), by applying 8 convolution units with a 9 x 9 kernel, 32 channels, and a stride of 2. Subsequently, for an \( I \times I \) pixel image, the primary capsules have a total of \( n_I \times n_I \times 32 \) 8D capsule outputs, where \( n_I = \text{cell}((256 - 36)/4) \). Each primary capsule output sees the output of all convolution units whose receptive fields overlap with the location of the center of the capsule. We first construct \( n_I = n_I \times n_I \times 32 \) primary capsule inputs \( t_i \in \mathbb{R}^8 \); subsequently, we obtain the primary capsule output \( u_i = \text{squash}(t_i) \), by applying the squash function, and the corresponding activity \( a_i = \frac{||t_i||^2}{1 + ||t_i||^2} \). Subsequently, the primary capsule outputs are propagated to the pruning layer, which selects the \( K \) most active capsules, forming \( U_K^{pri} \).
which we call the core primary capsules, and then the corresponding code vector \( c^{pri} \) is obtained. Based on \( c^{pri} \), we construct \( f_{j|i}(c^{pri}) \), \( p_j(c^{pri}) \), and \( f^{reg}_j(c^{pri}) \) using a combination of fully connected layers and convolution layers. More detailed structures are presented in Table 1. By applying those linear operators for propagation (i.e., \( f_{j|i}(c^{pri}) \) and \( p_j(c^{pri}) \)) and for regression (i.e., \( f^{reg}_{j}(c^{pri}) \)), we obtain the digit capsules and regressed primary capsules. As mentioned in Section 4, the number of digit capsules equals the number of labels, and we set the dimension of the digit capsules to 16. By computing the similarity between the core primary capsules and the regressed primary capsules using Eq.(4), the activity of each digit capsule is obtained, which is subsequently used to compute the loss in Section 4.

**5.2. Experiments on MNIST**

To evaluate the performance of the L-CapsNet, we performed two experiments. For the first experiment, we trained 60,000 images and tested 10,000 images of the 28 × 28 MNIST. For the second experiment, we trained 60,000 images of the 40 × 40 expanded MNIST, and tested 10,000 images of the affNIST. We assumed the same experimental environment (i.e., \( 9 \times 9 \) kernel on ReLuConv1 and PrimaryCaps) as in Sabour et al. (2017) on both experiments. This setting produces 1152 capsules for the MNIST experiment and 4608 capsules for the expanded MNIST and affNIST experiment. Furthermore, to consider various ratios of the number of selected capsules to the total capsules, we assumed more experimental settings with the kernel size \( 15 \times 15 \). In the MNIST experiment, we change the kernel size on PrimaryCaps from \( 9 \times 9 \) to \( 15 \times 15 \), and in the expanded MNIST and affNIST experiment, we set the kernel size \( 15 \times 15 \) on both layers. These settings produce 288 and 1152 primary capsules on MNIST and affNIST, respectively.

We performed the experiments with several values of \( K \) for pruning. Table 2 shows the test error for each case, together with the results of the CNN and CapsNet reported in Sabour et al. (2017).

Although the L-CapsNet resulted in higher test errors on MNIST classification, it dramatically outperformed on the affNIST test set. Recall that the expanded MNIST is con-
Table 2: Test error results of L-CapsNets

<table>
<thead>
<tr>
<th>Method</th>
<th>K</th>
<th>MNIST(%)</th>
<th>affNIST(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN (Sabour et al., 2017)</td>
<td>-</td>
<td>0.39</td>
<td>34.0</td>
</tr>
<tr>
<td>CapsNet (Sabour et al., 2017)</td>
<td>-</td>
<td>0.25</td>
<td>21.0</td>
</tr>
<tr>
<td>L-CapsNet (9 × 9 kernel)</td>
<td>50</td>
<td>0.74</td>
<td>13.0</td>
</tr>
<tr>
<td>L-CapsNet (9 × 9 kernel)</td>
<td>70</td>
<td>0.50</td>
<td>12.5</td>
</tr>
<tr>
<td>L-CapsNet (9 × 9 kernel)</td>
<td>100</td>
<td>0.80</td>
<td>13.2</td>
</tr>
<tr>
<td>L-CapsNet (15 × 15 kernel)</td>
<td>50</td>
<td>0.69</td>
<td>12.5</td>
</tr>
<tr>
<td>L-CapsNet (15 × 15 kernel)</td>
<td>70</td>
<td>0.73</td>
<td>12.2</td>
</tr>
<tr>
<td>L-CapsNet (15 × 15 kernel)</td>
<td>100</td>
<td>0.79</td>
<td>13.1</td>
</tr>
</tbody>
</table>

5.3. Analysis of the Effects of $K$

In the L-CapsNet, only $K$ most active capsules are used for classification tasks. Recall the example in Section 3.1 again, when the input is a human face, we hope that the core entities (e.g., nose, eyes, and mouth) are captured in $K$ most activate capsules, while capsules for representing irrelevant entities (e.g., wheels and mirrors) are deactivated. In other words, we hope that the L-CapNet clearly discriminates capsules for core entities versus irrelevant entities. We found from our experiments that this could be achieved by appropriately selecting the number of $K$.

We performed an analysis to see the effects of the hyperparameter $K$ in the L-CapsNet. Figure 4 illustrates the average activity levels of 300 most active primary capsules on the L-CapsNet over 5444 samples of digit “0” in the
MNIST training set for $K$ values of 50, 70, and 100. We can see that for low $K$ values (i.e., $K=50, 70$), the activity levels are extremely high or low. That is, the active and inactive capsules are distinguished clearly. This supports our adoption of $K$ motivated by the need for using only important capsules for robust performance. When $K = 100$, the distinction between active and inactive capsules is not as clear as the cases of $K=50$ and 70, maybe because it is not very effective in capturing disentangled entities and their poses using more capsules than actually needed. In contrast, when $K$ is low, each capsule may capture a disentangled entity and its pose, which results in clear discrimination between active and inactive capsules, which represent core and irrelevant entities, respectively. As Lenssen et al. (2018) pointed out the importance of disentangled representation for equivariance, for our experiments, low $K$ values would be preferred.

6. Conclusion

In this paper, we proposed a new building block of the capsule network that removes irrelevant capsules without losing information about the spatial relationship between lower-level and higher-level entities, based on our fining that only part of the entities (i.e., core entities) significantly contributes to capturing the part-whole spatial relationships. While the CapsNet captures the part-whole relationships by using an iterative routing-by-agreement, the L-CapsNet achieves the same goal by using both the pruning and ladder layers. More specifically, the pruning layer selects relevant lower-level capsules based on the activity level, using the fact that the activity level represents the probability of existence of an entity. In fact, we showed that highly active lower-level capsules tend to have large coupling coefficients for the desired parent (Figure 1) and that the L-CapsNet clearly discriminates relevant and irrelevant capsules using the activity level (Figure 4). Subsequently, the higher-level capsules can be constructed as a linear combination of the selected lower-level capsules. Unlike the CapsNet that takes the inner product between higher-level capsule and the prediction from the layer below as the “agreement” rule, the L-CapsNet takes how well lower-level capsule outputs are regressed from higher-level capsules as the agreement rule.

We select the $K$ most active capsules through the pruning layer, and subsequently propagate the activities of only the selected capsules into the layer above, from which we can expect the robust capability on the network. However, the method to determine the appropriate $K$ value according to the characteristics of data or tasks should be further studied. In our MNIST & affine MNIST experiments, the value of $K$ selected at a rate of approximately 6% of the total capsules produced the best results. Another important component of the L-CapsNet is the ladder layer. It is motivated by the reverse directional inference of the agreement/disagreement rule for the CapsNet. Based on our outperforming results on the affNIST data, we may conclude that the ladder layer provides the extrapolation capability or robustness on the L-CapsNet.

The reverse directional inference was similarly considered in Hinton et al. (2018). In their work, Hinton et al. (2018) approximate the mispredictions by higher-level capsules of lower-level capsule outputs by using the negative log probability density, avoiding the expensive matrix inversion of the prediction matrix $W_{ij}$. The L-CapsNet learns the inverse linear transformation $f_{reg}$ with a neural network rather than exactly inverting $W_{ij}$ or $f_{ji}$. For more extensive research, we plan to develop an algorithm for a bi-directional agreement/disagreement mechanism, based on the idea that we can predict not only higher-level capsule outputs from lower-level capsules, but can also regress lower-level capsule outputs from higher-level capsules. We expect this exchangeability to enable more robust capsule outputs to be extracted and improve the task performance. The code for L-CapsNet is available at https://github.com/taewonjeong/L-CapsNet.

Acknowledgements

The authors thank the reviewers for reviewing the paper and providing valuable comments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2018R1C1B6004511).

References


Dhillon, G. S., Azizzadenesheli, K., Lipton, Z. C., Bernstein, J., Kossaifi, J., Khanna, A., and Anandkumar, A.


