Report: 3D Point Capsule Network with EM routing

Christopher Murray and Guillaume Dufau

January 2020

1 Introduction

Significant advances in image and point cloud processing tasks have been made in recent years due to the development of Deep learning techniques such as Convolutional Neural Networks (CNN) and PointNet. Despite increasingly good results, these techniques suffer from the lack of interpretability and control. Recently, a new approach called Capsule Networks has emerged showing promising results, and already outperforms CNNs in some tasks. There is a lot of interest in developing this technique as it has shown superior performance to CNNs, and also offers more interpretability. Our project focuses on further developing the technique's usage as an encoder/decoder architecture for the point cloud processing network known as the 3D Point Capsule Network [Zhao et al., 2019a]. Followed by the study of the principles of CapsNets for images and 3D objects we attempted to implement some adjustment to the idea used in [Zhao et al., 2019a]. We updated the routing algorithm used by the 3D Point Capsule Network, which was previously only used for CapsNets for images. The report is then decomposed into two parts: our understanding of the current main ideas of Capsule Networks in general case, and a review of the 3D cloud point idea plus an improvement to the routing algorithm we implemented.

2 Capsule Networks Routing-by-Agreement

The first Capsule Network paper to receive huge interest from the field is Dynamic Routing Between Capsules [Sabour et al., 2017]. In this version the routing by agreement procedure fulfills the same function as the max pooling layers (transfer local information from lower layers to higher) from CNNs, but does so in a more intelligent way so there is not a trade-off between information resolution and locality. A Capsule Network stacks layers of capsules and each layer represent a certain hierarchy of features. Each capsule is composed of vectors which contain the pose information of their features, and the likelihood of a given feature is determined through the length of its pose vector. The difference between convolution & pooling with capsules comes from the equivariance. An image with, from top to bottom, a mouth, two eyes and a nose will be classified as face by an usual CNNs whereas the capsule would understand its more likely a Picasso than a face. Due to the instantiating parameters encapsulating the poses, the spatial information is transmitted between different hierarchical features. Therefore CapsNets are more efficient for certain task such as segmentation with overlapping objects than CNNs. And it has been shown that they require less training example to converge with better results.

2.1 Dynamic Routing

Inside a Capsule Network, it's the succession of capsules layers and their intercommunication that allow the equivariance of the entire network. A first convolution layer is applied over the input and its output is simply reshaped into a layer called PrimaryCaps composed of capsules (the lower level features). It can communicate with a third capsules layer say ClassCaps through a **voting procedure**. In this section we focus on the routing by agreement from [Sabour et al., 2017]. Each capsule is a vector which can have different sizes between layers. A capsule represents a feature. Each capsule is a vector with instantiation parameters representing the pose of a feature and its length represents the likelihood of presence of the feature. The figure 1 shows an intuition of how the variants are transmitted along the Network.

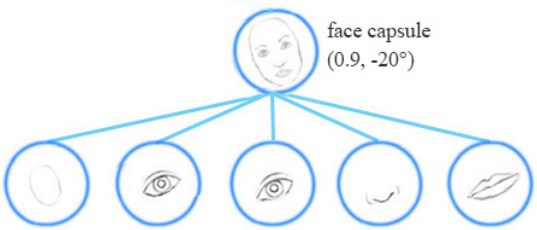

Figure 1: How a capsule can detect the variants. Source: Jonathan Hui

Inside a standard Neural Network (NN) the information between layers is transmitted using a scalar. But for CapsNets the connection concerns two vectors (or Matrices). This connection is called a **transformation matrix**. The **routing** iterations (by agreement or later EM) enables the so called 'part-whole' relationship and information transmission between layers. In figure 1 the light blue connections are in fact transformation matrices + vote coefficients from the routing iterations. And the parts (mouth, eye etc) can be connected to the whole (face in this example, or many more inside a same layer).

Now take the example of digit recognition. As show in figure 2, the layer (l) consists of six features (six capsules) and (l+1) has two higher level features.

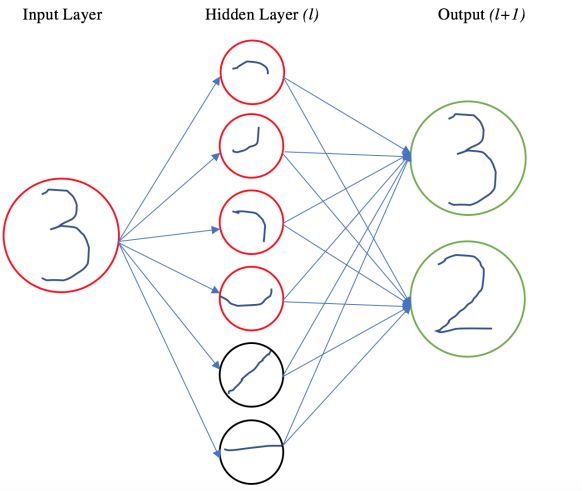

Figure 2: Each "node" is a capsule i.e. a vector and the arrows are the transformation matrices. Source: Sahaj Garg

The features in (l) come from the (l+1) objects (or features if many more layers). The routing algorithm figures out the probability that a feature in (l) is present according to the (l+1) activations. Indeed if many of the features in (l) are agreeing towards the 3 (same rotation and correct position of all the lower level features) then it activates the 3 capsule and encode the same rotation as the lower level features. The idea of 'vote' is here. Each (l) feature can vote for the (l+1) it thinks it belongs to.

How is this vote procedure computed mathematically ? The j^{th} capsule in (l+1) is computed with:

$$\hat{u}_{j|i} = W_{ij} u_i$$

$$s_j = \sum_i c_{ij} \hat{u}_{j|i}$$

W_{ij} is the transformation matrix learned during backpropagation and c_{ij} are the coupling coefficients learnt during the dynamic routing process. It can be seen as the likelihood of i activating j. Since the process works with vectors there is no ReLu function but rather a squashing function that normalizes the capsules. The final value of the j^{th} capsule in (l+1) is therefore:

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|}$$

$\hat{u}_{j|i}$ is the vote of i to j. If $\hat{u}_{j|i}$ and v_j are two close vectors then i and j are related. The paper introduces:

$$b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$$

This b coefficient is updated through multiple iterations in the dynamic routing (usually 3 in the paper). Finally the c_{ij} can be computed using a softmax over the b_{ij}:

$$c_{ij} = \frac{\exp b_{ij}}{\sum_k \exp b_{ik}}$$

Procedure 1 Routing algorithm.

procedure ROUTING($\hat{\boldsymbol{u}}_{j|i}$, 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: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
    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 $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
    for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
  return $\mathbf{v}_j$

Figure 3: The updating procedure in between two caps layers

The entire Network is composed of multiple types of Capsules Layers. The PrimaryCaps layer is created through a more traditional convolution layer (without pooling). In this paper [Sabour et al., 2017] the PrimaryCaps are transformed into ClassCaps directly, using the routing process above. And the resulted ClassCaps layer can be passed into a MLP to extract the class prediction in this case. See figure below.

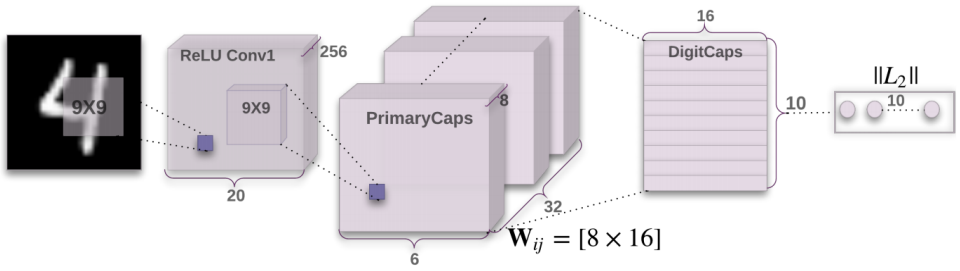

Figure 4: A first convolution layer reshaped into PrimaryCaps, the routing algorithms produces the DigitCaps or ClassCaps layer which is passed into a MLP

2.2 An improvement: Matrix capsules with Expectation Maximization (EM) routing

Hinton et al. published a second paper, Matrix Capsules with EM Routing [Hinton et al., 2018]. Here the capsules are not vectors anymore but rather matrices (4 by 4) + an activation scalar, figure 5.

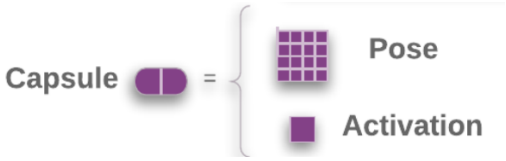

Figure 5: In the EM routing algorithm, the capsules are represented as matrices and each capsule has an activation scalar

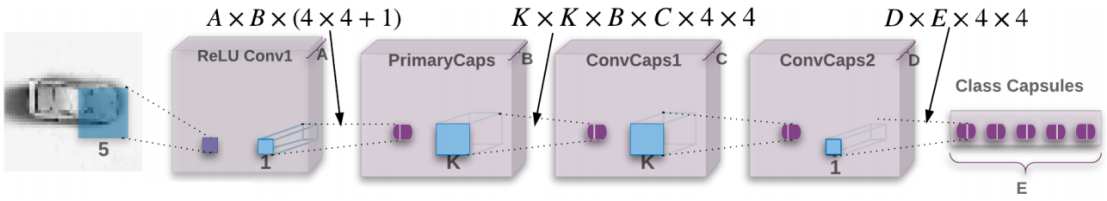

Figure 6: A convolution layer creates the PrimaryCaps followed by two latent caps layers derived using EM routing procedure

The whole network in this paper is summarized in figure 6. Contrary to the previous Network there are multiple Latent Capsules Layers called ConvCaps. The objective of the EM algorithm is to recreate the part-whole relationship between layers of capsules. From the figure below the idea of vote comes from the clustering.

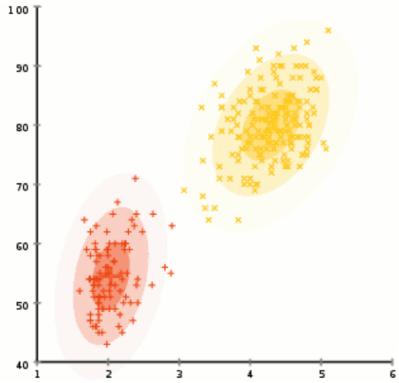

Figure 7: Each point can represent one (l) layer vector and the clusters are the votes of the (l) capsules.

This is also an iterative process, starting from two or more Gaussians it update their center and Covariance matrices to converge towards a convenient cluster. In the case of CapsNets, the vote v_{ij} from i^{th} capsule in (l) is computed using a pose matrix M_i and a transformation matrix W_{ij} which is again learnt through backpropagation:

$$v_{ij} = M_i W_{ij}$$

The poses (matrices) and activations of the (l+1) layer will be computed by taking v_{ij} and a_i as input into the EM routing algorithm. To compute the degree of belonging of one small feature to a whole, the EM algorithm provides an assignment probabilities r_{ij}.

The EM algorithm represents the pose matrices as parameters for the Gaussian models, that is, each μ in the Gaussian represents an element of the pose in the end. An interesting further exploration could be to use the σ from the Gaussian to get the output, even-though the pose will indirectly depend on the sigmas as we'll see.

Since each v_{ij} represents is a 16 elements matrix, note v_{ij}^h the h^{th} component. As explained above, during the procedure each vote from i to j (v_{ij}) follows a 16 dimensional Gaussian model. Therefore to compute the probability of the vote v_{ij}^h belonging to the j's Gaussian model we get:

$$p_{i|j}^h = \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} \exp\left(-\frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}\right)$$

Taking the log off this formula, it is now possible to estimate the likelihood of a capsule being activated thanks to a formula cold the cost. If the cost if high the vote does not match the (l+1) Gaussian distribution. The idea of the cost computation is as follows:

$$cost_j^h = \sum_i r_{ij} cost_{ij}^h$$

And finally to better estimate whether a capsule j on (l+1) will be activated or not, a_j describes a relative importance of the votes (or non-votes) from the (l) level capsules assigned to the j_{th} capsule's 'description cost' in (l+1). This cost is noted β_j and is learned through backpropagation. This ratio can be seen as the number of data points (capsules) needed to describe the j_{th} feature.

$$a_j = logistic(\lambda(b_j - \sum_h cost_j^h))$$

The λ parameter has to be manually fixed. To sum up, the values r, μ, σ and a_j of each part-whole relation is computed during the EM routing occurring at each CapsNet usage. Here, we only went through the main ideas and intuitions. Since going in pure mathematical details would mostly be paraphrasing, further explanation can be found in [Hinton et al., 2018] or in the Jonathan Hui's blog post.

3 Implementation of the paper for 3D point clouds

3.1 Algorithm

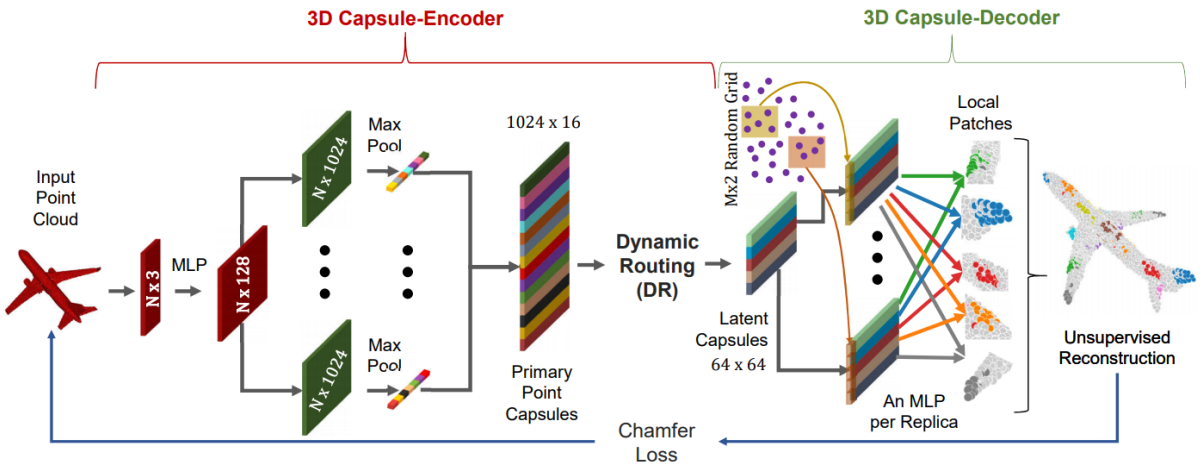

Figure 8: From standard input point cloud to modified reconstruction using capsules to encode the important features

The figure 8 summarizes the steps of 3D Point Capsule Networks [Zhao et al., 2019a]. The idea is to use a encoder-decoder structure. Latent capsules are obtained from generic point clouds. This latent space is parameterizing a low dimensional vector giving explicit control on the basis functions composing the reconstructed 3D shape thanks to the capsules. From a single generic plane shape this procedure enable to create elongate for example the length of the wings with a simple capsule modification in the latent space. The authors provide other interesting possibilities from this techniques in an extended paper [Zhao et al., 2019b]. But this project focuses on the encoding procedure of the original paper, left side in 8.

The **encoding** approach is decomposed as follows:

- From a list of points (Nx3), extract information with a multi-layer perception (MLP) into N vectors of size 128 (Nx128);
- From the new vectors, uses a CNN layer to finally create the Primary Caps layer described above (here a different shape with 1024x16);
- The dynamic routing can start, from the primaryCaps it creates a latent capsule layer (64x64) which will be the starting point of the decoding part and potential modification.

Since part of the Network has to be trained through backpropagation, in the 3D point clouds CapsNet the author use a loss to compare the input of the network and output point cloud, the Chamfer distance (between two point clouds):

$$D_{chamfer}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

3.2 Our improvement: using matrix capsules and EM routing in the 3D point caps

The original paper [Zhao et al., 2019a] uses a routing-by-agreement procedure from [Sabour et al., 2017] described above to create the latent capsules layer. This procedure is well suited to this problem since it already provides good results when the PrimaryCaps are in vector format. As it turns out, the new EM routing seems to provide better results for most of the cases and requires less time to be trained (number of examples as well as real running time) which is a real pain point in the case of 3D point applications. Therefore this part of the report highlights our improvement over the procedure which was to use EM routing and matrix capsules for the encoding part of the 3DCapsNet.

When it comes to encoding pointcloud information, there is no need to keep going with a 2D convolution at each layer. Indeed, the spatial information is intrinsically encoded in each of the vectors at the beginning (Nx128) which is not the case when working with an image. From figure 6 the new procedure stills uses the (1024xN) vector as a PrimaryCaps. To go from a PrimaryCaps to a Convcaps (or latent capsules layer) the EM algorithm requires a 2D convolution. But as said above, the spatial information that one could found in an image and its 2D Conv is not needed anymore. In the end it suffices to use a 2D conv and considering that the 1024xN vector is an image with just one pixel and apply a 1x1 kernel. This little trick allow the use of the classical EM routing, and as it turns out the results ended being equivalent (without tuning the routing algorithm) with a faster convergence. Due to a lack of time regarding the slow celerity of the training, we couldn't tune some of the EM parameters or even increase the number of ConvNets layers which we think can potentially improve the results.

3.3 Our results

The following loss curves from figure 9 correspond to training the 3D Point Capsule Network using either the original Dynamic Routing by Agreement algorithm, and our new implementation using the Expectation Maximization routing algorithm. One thing we noticed is the difference in training time to achieve relatively similar results. Training using Dynamic Routing by Agreement took **55 minutes** for 3 epochs, while using Expectation Maximization Routing took **45 minutes** for 3 epochs. Another point of consideration is that there was no tuning done of the EM algorithm, which Hinton noted in the original EM paper, as being a large part of the success of the algorithm over the original Dynamic Routing by Agreement.

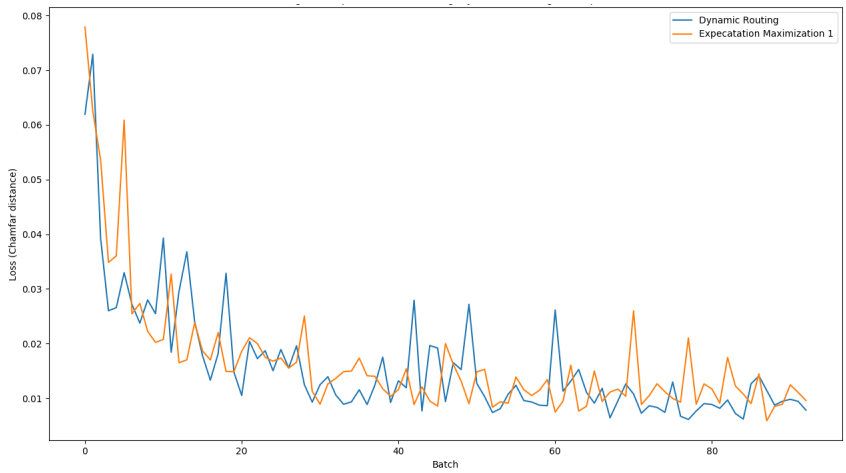

Figure 9: Loss curves of training the 3D Point Caps net using both Dynamic Routing by Agreement and Expectation Maximization routing

4 Conclusion

For this project, we came to understand the inspiration and mechanics of Capsule Networks, as well as to implement a newer routing algorithm to a state of the art point cloud processing Capsule Network. The main challenges we faced during this project were first, really understanding the mechanics of Capsule networks, and then implementing the Expectation Maximization Routing algorithm. The EM algorithm was especially difficult due to the novelty of Capsule Networks means that the current Deep Learning libraries do not natively support them yet, so modifying an existing implementation to support 1-dimensional Convolutions was near impossible. The results we found using the EM routing were that the training process was 20 percent faster, and without tuning any parameters of the EM routing, delivered similar training convergence as using the original routing algorithm.

References

[Hinton et al., 2018] Hinton, G., Sabour, S., and Frosst, N. (2018). Matrix capsules with em routing.

[Sabour et al., 2017] Sabour, S., Frosst, N., and Hinton, G. (2017). Dynamic routing between capsules.

[Zhao et al., 2019a] Zhao, Y., Birdal, T., Deng, H., and Tombari, F. (2019a). 3d point capsule networks.

[Zhao et al., 2019b] Zhao, Y., Birdal, T., Deng, H., and Tombari, F. (2019b). 3d point capsule networks supplementary material.