

# Dynamic Routing Between Capsules

Authors: Sara Sabour, Nicholas Frosst, Geoffrey E Hinton

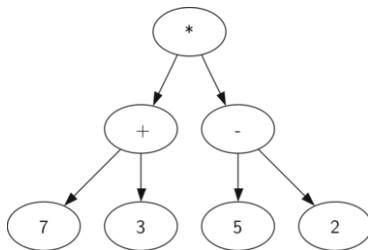
November 2017

# Introduction

- A capsule is a group of neurons whose activity vector represents an entity - either an object or an object part
- The length of the vector represents the probability that a certain entity exists
- Achieves state-of-the-art performance on MNIST and is better at recognizing highly overlapping digits than a convolutional network

## Reference with human vision

- The human vision processes only a tiny fraction of the optic array at the highest resolution
- Assumption that the visual system creates a parse tree on each fixation



(a) Parse tree example:  
 $(7 + 3) * (5 - 2)$

# Capsules as parse trees

- A parse tree can be represented by a multilayer neural network
- Each layer will be divided into many capsules
- Using an iterative routing process, each capsule will choose a capsule in the layer above to be its parent in the tree - this process solves the problem of assigning parts to wholes

# Describing the capsule

- The vector of activities of a capsule represent the various properties of a particular entity that is present in the image (pose, deformation, velocity, texture etc.)
- The existence of an object is defined by using the overall length of the activity vector of a capsule - a non-linearity is applied so that the length of the vector output cannot exceed 1
- A lower level capsule is assigned to a higher level capsule by a “routing-by-agreement” algorithm

# How the vector inputs and outputs of a capsule are computed

The length of the output vector of a capsule should represent the probability that the entity represented by the capsule is present in the current input. A non-linear squashing function is used to ensure that short vectors get shrunk to almost zero length and long vectors get shrunk to a length slightly below 0:

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

where  $v_j$  is the vector output of capsule  $j$  and  $s_j$  is its total input.

# How the vector inputs and outputs of a capsule are computed

For all but the first layer of capsules, the total input to a capsule  $s_j$  is a weighted sum over all prediction vectors  $\hat{u}_{j|i}$  from the capsules in the layer below and is produced by multiplying the output  $u_i$  of a capsule in the layer below by a weight matrix  $W_{ij}$ :

$$s_j = \sum_i c_{ij} \hat{u}_{i|j}, \quad \hat{u}_{i|j} = W_{ij} u_i \quad (2)$$

where the  $c_{ij}$  are coupling coefficients that are determined by the iterative dynamic routing process.

# How the vector inputs and outputs of a capsule are computed

The coupling coefficients between capsule  $i$  and all the capsules in the layer above sum to 1 and are determined by a “routing softmax” whose initial logits  $b_{ij}$  are the log prior probabilities that capsule  $i$  should be coupled to capsule  $j$ :

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \quad (3)$$

The initial coupling coefficients are iteratively refined by measuring the agreement between the current output  $v_j$  of each capsule,  $j$ , in the layer above and the prediction  $\hat{u}_{j|i}$  made by capsule  $i$ .



# How the vector inputs and outputs of a capsule are computed

The agreement is simply the scalar product  $a_{ij} = v_j \cdot \hat{u}_{j|i}$ . This agreement is treated as if it were a log likelihood and is added to the initial logit,  $b_{ij}$  before computing the new values for all the coupling coefficients linking capsule  $i$  to higher level capsules.

# Routing algorithm

```
1: function ROUTING( $\hat{u}_{j|i}, r, l$ )
2:   for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow 0$ .
3:   for  $r$  iterations do
4:     for all capsule  $i$  in layer  $l$ :  $c_i \leftarrow \text{softmax}(b_i)$ 
5:     for all capsule  $j$  in layer  $(l + 1)$ :  $s_j \leftarrow \sum_i c_{ij} \hat{u}_{j|i}$ 
6:     for all capsule  $j$  in layer  $(l + 1)$ :  $v_j \leftarrow \text{squash}(s_j)$ 
7:     for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :
8:        $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$ 
   return  $v_j$ 
```

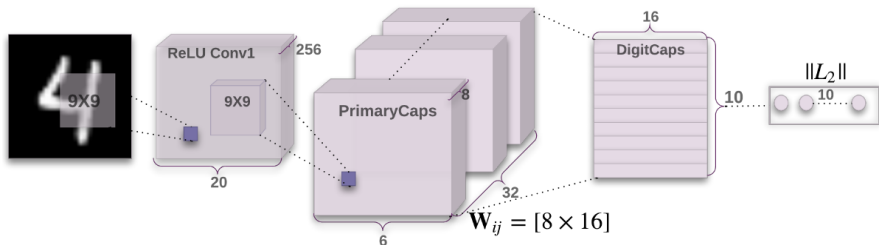
## Margin loss for digit existence

Because the length of the instantiation vector is used to represent the probability that a capsule's entity exists, the top-level capsule for digit class  $k$  should have a long vector if and only if that digit is present in the image. For multiple digits, a margin loss is used:

$$L_c = T_c \max(0, m^+ - \|v_c\|)^2 + \lambda(1 - T_c) \max(0, \|v_c\| - m^-)^2 \quad (4)$$

where  $T_c = 1$  iff a digit of class  $c$  is present and  $m^+ = 0.9$  and  $m^- = 0.1$ . The  $\lambda$  down-weighting of the loss for absent digit classes stops the initial learning from shrinking the lengths of the activity vectors of all the digit capsules. The total loss is simply the sum of the losses of all digit capsules.

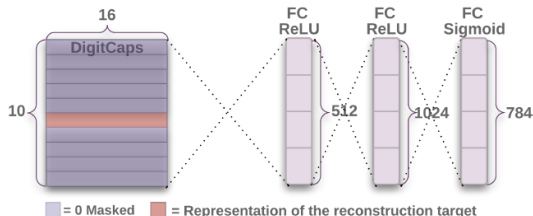
# CapsNet architecture



(b) CapsNet architecture

- Shallow network with only two convolutional layers and one fully connected layer
- Routing is implemented only between PrimaryCaps and DigitCaps layers













# Reconstruction as a regularization method



(c) Reconstruction from a capsule representation

An additional reconstruction loss is used to encourage the digit capsules to encode the instantiation parameters of the input digit. The output of the correct digit capsule is fed into a decoder consisting of 3 fully connected layers that model the pixel intensities. The objective function is the sum of squared differences between the outputs of the logistic units and the pixel intensities. This loss is scaled by 0.0005 so that it does not dominate the margin loss during training.

# Reconstruction results

$(l, p, r)$	$(2, 2, 2)$	$(5, 5, 5)$	$(8, 8, 8)$	$(9, 9, 9)$		$(5, 3, 5)$	$(5, 3, 3)$
Input							
Output							

(d) Digits reconstructed from the output of a capsule

## Results on MNIST


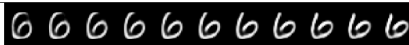



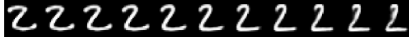
Method	Routing	Reconstruction	MNIST (%)	MultiMNIST (%)
Baseline	-	-	0.39	8
CapsNet	1	no	$0.34_{\pm 0.032}$	-
CapsNet	1	yes	$0.29_{\pm 0.011}$	7
CapsNet	3	no	$0.35_{\pm 0.036}$	-
CapsNet	3	yes	<b><math>0.25_{\pm 0.005}</math></b>	<b>5</b>

(e) Comparisons between different network configurations and a CNN baseline

The baseline is a standard CNN with three convolutional layers of 256, 256, 128 channels. Each has 5x5 kernels and stride of 1. The last convolutional layer is followed by two fully connected layers of size 328, 192. The last fully connected layers is connected with dropout to a 10 class softmax layer with cross entropy loss.

# What the individual dimensions of a capsule represent

- The dimensions of a digit capsule should learn to span the space of variations in the way digits of that class are instantiated (stroke thickness, skew and width)
- To see what the individual dimensions represent, a perturbed output vector can be fed to the decoder network to see how the perturbation affects the reconstruction

Scale and thickness	
Localized part	
Stroke thickness	
Localized skew	
Width and translation	
Localized part	

(f) Dimension perturbations



# Robustness to Affine Transformations

- Experiments show that each DigitCaps capsule learns a more robust representation for each class than a traditional convolutional network
- To test the robustness of CapsNet to affine transformations, a training was done on a padded and translated MNIST set, in which each example is an MNIST digit placed randomly on a black background
- The network was tested on the affNIST<sup>1</sup> data set, in which each example is an MNIST digit with a random small affine transformation
- CapsNet achieved 79% accuracy, while a traditional convolutional model with a similar number of parameters only 66%

---

<sup>1</sup>Available at <http://www.cs.toronto.edu/~tijmen/affNIST/>.

# Segmenting highly overlapping digits

- Dynamic routing can be viewed as a parallel attention mechanism that allows each capsule at one level to attend to some active capsules at the level below and to ignore others; This should allow the model to recognize multiple object in the image even if objects overlap
- To test this assumption, the MultiMNIST dataset was created by overlaying a digit on top of another from the same set, but different class, having 80% average overlap