

# Oh! Capsule

---

date: 2017/11/09, author: wu

## 1 Transforming Auto-encoder(ICANN 2011 )

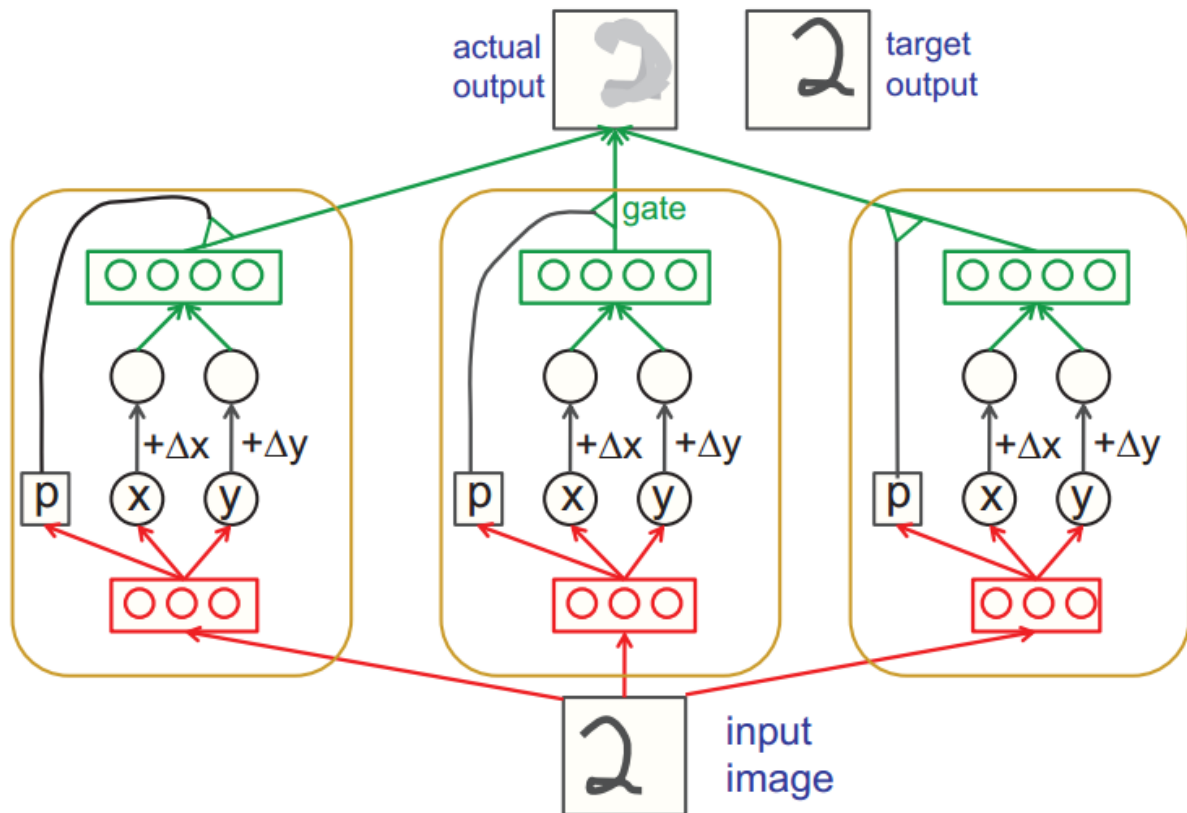
---

### 1.1 Introduction

Artificial neural networks should use local “capsules” that perform some quite complicated internal computations on their inputs and then encapsulate the results of these computations into a small vector of highly informative outputs. Each capsule learns to recognize an implicitly defined visual entity over a limited domain of viewing conditions and deformations and it outputs both the probability that the entity is present within its limited domain and a set of “instantiation parameters” that may include the precise pose, lighting and deformation of the visual entity relative to an implicitly defined canonical version of that entity. When the capsule is working properly, the probability of the visual entity being present is locally invariant – it does not change as the entity moves over the manifold of possible appearances within the limited domain covered by the capsule. The instantiation parameters, however, are “equivariant” – as the viewing conditions change and the entity moves over the appearance manifold, the instantiation parameters change by a corresponding amount because they are representing the intrinsic coordinates of the entity on the appearance manifold.

### 1.2 Details about a image shift example

#### 1. Model graph



## 2. Core codes (use tensorflow)

```
def build(self, X_in, extra_in):
    rec = tf.sigmoid(self.fc_layer(X_in, self.in_dim, self.r_dim,
                                   'recog_layer_pre_act'), 'recog_layer')

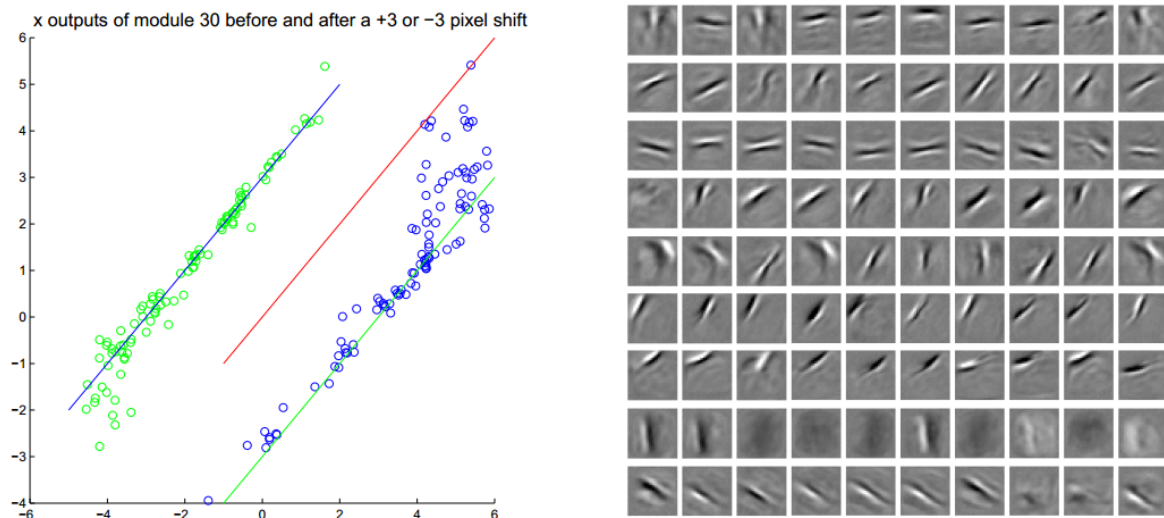
    xy_vec = self.fc_layer(rec, self.r_dim, 2, 'xy_prediction')
    pro = tf.sigmoid(self.fc_layer(rec, self.r_dim, 1,
                                   'probability_lin'), 'probability_prediction')
    probability_vec = tf.tile(pro, (1, self.in_dim))

    xy_extend = tf.add(xy_vec, extra_in)
    gen = tf.sigmoid(self.fc_layer(xy_extend, 2, self.g_dim, 'gen_pre_act'),
                     'gen_layer')

    out = self.fc_layer(gen, self.g_dim, self.in_dim, 'out_prediction')

    return tf.multiply(out, probability_vec)
```

## 3. Experiment Result



**Fig. 2. Left:** A scatterplot in which the vertical axis represents the x output of one of the capsules for each digit image and the horizontal axis represents the x output from the same capsule if that image is shifted by +3 or -3 pixels in the x direction. If the original image is already near the limit of the x positions that the capsule can represent, shifting further in that direction causes the capsule to produce the wrong answer, but this does not matter if the capsule sets its probability to 0 for data outside its domain of competence. **Right:** The outgoing weights of 10 of the 20 generative units for 9 of the capsules.

### 1.3 Work to do

Once pixel intensities have been converted into the outputs of a set of active, first-level capsules each of which produces an explicit representation of the pose of its visual entity, it is relatively easy to see how larger and more complex visual entities can be recognized by using **agreements of the poses** predicted by active, lower-level capsules.

**Agreements of the poses example:** If a capsule can learn to output the pose of its visual entity in a vector that is linearly related to the “natural” representations of pose used in computer graphics, there is a simple and highly selective test for whether the visual entities represented by two active capsules, A and B, have the right spatial relationship to activate a higher-level capsule, C. Suppose that the pose outputs of capsule A are represented by a matrix,  $T_A$ , that specifies the coordinate transform between the canonical visual entity of A and the actual instantiation of that entity found by capsule A. If we multiply  $T_A$  by the part-whole coordinate transform  $T_{AC}$  that relates the canonical visual entity of A to the canonical visual entity of C, we get a prediction for  $T_C$ . Similarly, we can use  $T_B$  and  $T_{BC}$  to get another prediction. If these predictions are a good match, the instantiations found by capsules A and B are in the right spatial relationship to activate capsule C and the average of the predictions tells us how the larger visual entity represented by C is transformed relative to the canonical visual entity of C. If, for example, A represents a mouth and B represents a nose, they can each make a prediction for the pose of the face. If these predictions agree, the mouth and nose must be in the right spatial relationship to form a face. An interesting property of this way of performing shape recognition is that the knowledge of part-whole relationships is viewpoint-invariant and is represented by weight matrices whereas the knowledge of the instantiation parameters of currently observed objects and their parts is viewpoint-equivariant and is represented by neural activities

## 2 Dynamic Routing Between Capsules(NIPS 2017)

## 2.1 Introduction

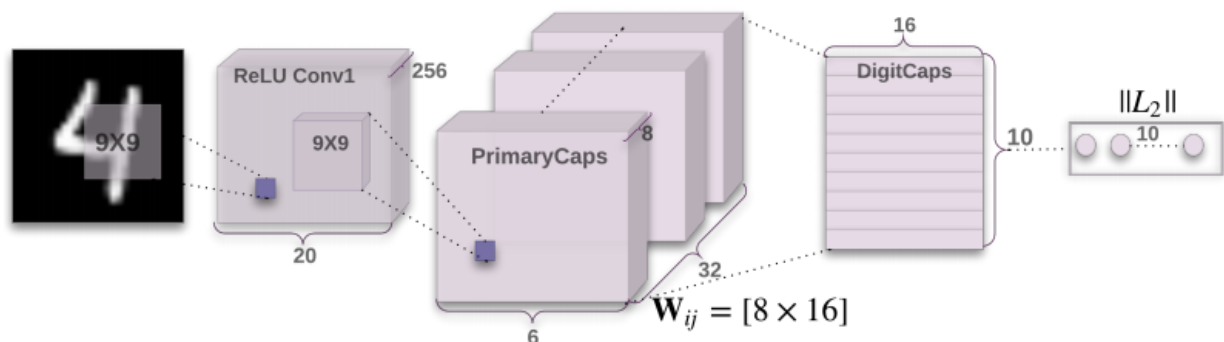
The previous paper said that: "Once pixel intensities have been converted into the outputs of a set of active, first-level capsules each of which produces an explicit representation of the pose of its visual entity, it is relatively easy to see how larger and more complex visual entities can be recognized by using **agreements of the poses** predicted by active, lower-level capsules. " But it didn't do the work to build a multilayer capsules network.

This paper give a implementation of the multilayer capsules network that do classification work on **MNIST** dataset. And it give a routing algorithm comparing to just the word **agreements**. It give a detail formulation to measure **agreements**. Then it use **Margin Loss** and **Reconstruction** as a regularization method

Besides, instead of using common full connection layer to initialize the first-level capsules, it use convolutional neural network to do this work. The paper says: "Convolutional neural networks (CNNs) use translated replicas of learned feature detectors and this allows them to translate knowledge about good weight values acquired at one position in an image to other positions. This has proven extremely helpful in image interpretation. Even though we are replacing the scalar-output feature detectors of CNNs with vector-output capsules and max-pooling with routing-by-agreement, we would still like to replicate learned knowledge across space, so we make all but the last layer of capsules be convolutional. "

## 2.2 Model detail

### 1. Model graph



### 2. Core codes and algorithms

1. First layer is a common convolutional layer
2. Second layer is PrimaryCaps layer. But One can see ist as a Convolution layer with  $v_j = \frac{|s_j|^2}{1 + |s_j|^2} \frac{s_j}{|s_j|}$  as its block non-linearity. Just as the following codes.(In the following codes, squash is a function to implement the former eq. )

```
def PrimaryCap(inputs, dim_vector, n_channels, kernel_size, strides, padding):  
    outputs = []  
    output = layers.Conv2D(filters=dim_vector*n_channels, kernel_size=kernel_size,  
                           strides=strides, padding=padding)(inputs)  
    outputs = layers.Reshape(target_shape=[-1, dim_vector])(output)  
    return layers.Lambda(squash)(outputs)
```

3. In total PrimaryCapsules has [32; 6; 6] capsule outputs (each output is an 8D vector) . (shape is [32\*6\*6, 8]). The third layer is Digit layer. Between these two layers, the author use the routing algorithm as bellow.

---

**Procedure 1** Routing algorithm.

---

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

---

#### 4. Margin loss.

We are using the length of the instantiation vector to represent the probability that a capsule's entity exists, so we would like the top-level capsule for digit class  $k$  to have a long instantiation vector if and only if that digit is present in the image. To allow for multiple digits, we use a separate margin loss,  $L_k$  for each digit capsule,  $k$ :

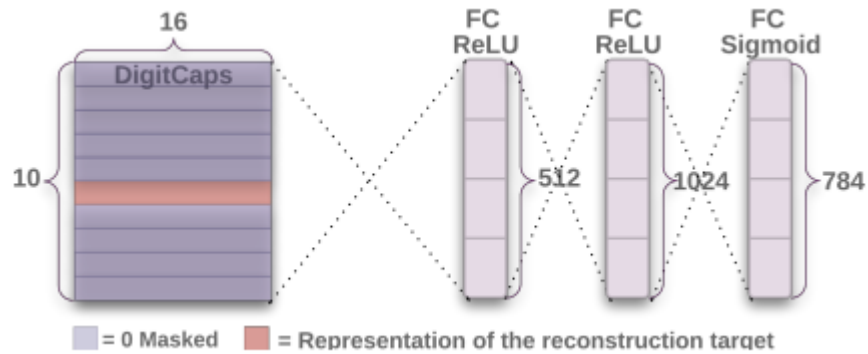
$$L_c = T_c \max(0, m^+ - \|\mathbf{v}_c\|)^2 + \lambda (1 - T_c) \max(0, \|\mathbf{v}_c\| - m^-)^2 \quad (4)$$

where  $T_c = 1$  iff a digit of class  $c$  is present<sup>3</sup> 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. We suggest  $\lambda = 0.5$ . The total loss is simply the sum of the losses of all digit capsules.

#### 5. Reconstruction as a regularization method.

We use an additional reconstruction loss to encourage the digit capsules to encode the instantiation parameters of the input digit. During training, we mask out all but the activity vector of the correct digit capsule. Then we use this activity vector to reconstruct. The output of the digit capsule is fed into a decoder consisting of 3 fully connected layers that model the pixel intensities as described in Fig. 2. We minimize the sum of squared differences between the outputs of the logistic units and the pixel intensities. **We scale down this reconstruction loss by 0.0005** so that it does not dominate the margin loss during training. As illustrated in Fig. 3 the reconstructions from the 16D output of the CapsNet are robust while keeping only important details.

Fig. 3:



### 3. Experiment Result

#### 1. error rate on MNIST

Table 1: CapsNet classification test accuracy. The MNIST average and standard deviation results are reported from 3 trials.


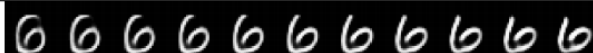




Method	Routing	Reconstruction	MNIST (%)	MultiMNIST (%)
Baseline	-	-	0.39	8
CapsNet	1	no	$0.34 \pm 0.032$	-
CapsNet	1	yes	$0.29 \pm 0.011$	7
CapsNet	3	no	$0.35 \pm 0.036$	-
CapsNet	3	yes	<b><math>0.25 \pm 0.005</math></b>	<b>5</b>

## 2. What the individual dimensions of a capsule represent.

Since we are passing the encoding of only one digit and zeroing out other digits, the dimensions of a digit capsule should learn to span the space of variations in the way digits of that class are instantiated. These variations include stroke thickness, skew and width. They also include digit-specific variations such as the length of the tail of a 2. We can see what the individual dimensions represent by making use of the decoder network. After computing the activity vector for the correct digit capsule, we can feed a perturbed version of this activity vector to the decoder network and see how the perturbation affects the reconstruction. Examples of these perturbations are shown in Fig. 4. We found that one dimension (out of 16) of the capsule almost always represents the width of the digit. While some dimensions represent combinations of global variations, there are other dimensions that represent variation in a localized part of the digit. For example, different dimensions are used for the length of the ascender of a 6 and the size of the loop

Fig 4:

Figure 4: Dimension perturbations. Each row shows the reconstruction when one of the 16 dimensions in the DigitCaps representation is tweaked by intervals of 0.05 in the range  $[-0.25, 0.25]$ .

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

## 3. Robustness to Affine Transformations

To test the robustness of CapsNet to affine transformations we trained a CapsNet and a traditional convolutional network (with MaxPooling and Dropout) on a padded and translated MNIST training set, in which each example is an MNIST digit placed randomly on a black background of  $40 \times 40$  pixels. We then tested this network on the affNIST4 data set, in which each example is an MNIST digit with a random small affine transformation.

Net\Accuracy	Expanded MNIST Test Set	Affined Test Set
CapsNet	99.23%	79
CNN	99.22	66

#### 4. Segmenting highly overlapping digits

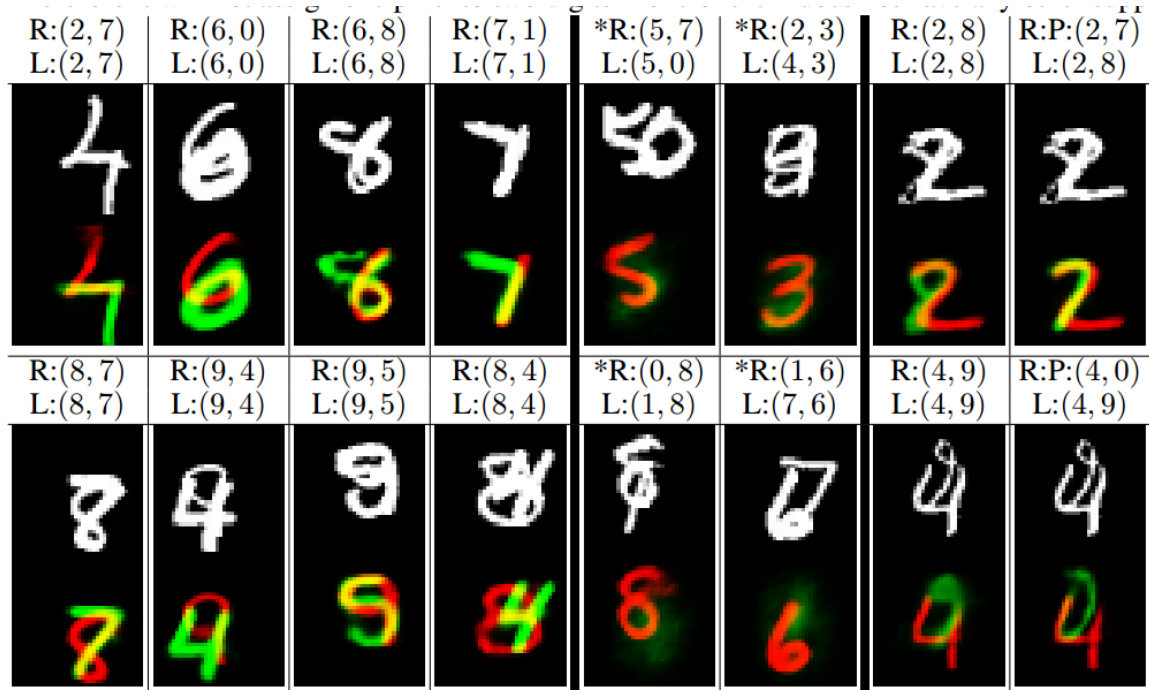


Figure 5: Sample reconstructions of a CapsNet with 3 routing iterations on MultiMNIST test dataset. **The two reconstructed digits are overlaid in green and red as the lower image.** The upper image shows the input image. L:(l1; l2) represents the label for the two digits in the image and R:(r1; r2) represents the two digits used for reconstruction. The two right most columns show two examples with wrong classification reconstructed from the label and from the prediction (P). In (2; 8) example the model confuses 8 with a 7 and in (4; 9) it confuses 9 with 0. The other columns have correct classifications and show that model accounts for all the pixels while being able to assign one pixel to two digits in extremely difficult scenarios (column 1 - 4). Note that in dataset generation the pixel values are clipped at 1. The two columns with **the (\*) mark** show reconstructions from a digit that is neither the label nor the prediction.

## 2.3 Drawbacks

One drawback of Capsules which it shares with generative models is that it likes to account for everything in the image so it does better when it can model the clutter than when it just uses an additional “orphan” category in the dynamic routing. In CIFAR-10, the backgrounds are much too varied to model in a reasonable sized net which helps to account for the poorer performance.

## 3 Matrix capsules with EM routing (ICLR 2018, under review)



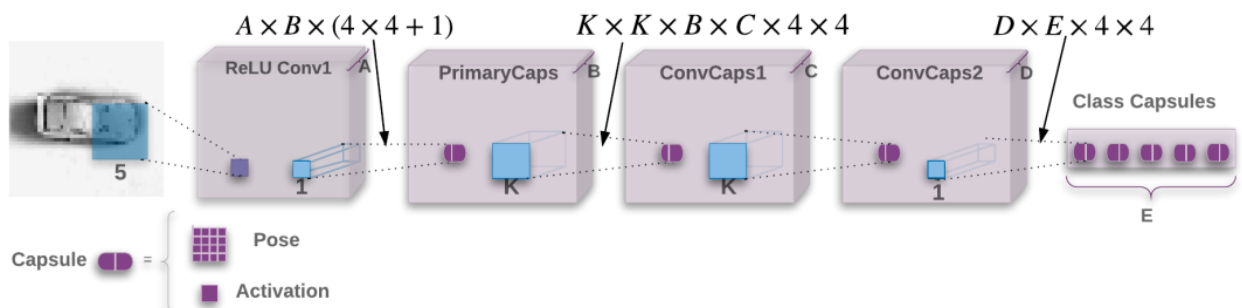
## 3.1 Introduction

The system in previous paper has several deficiencies as arguing in this paper:

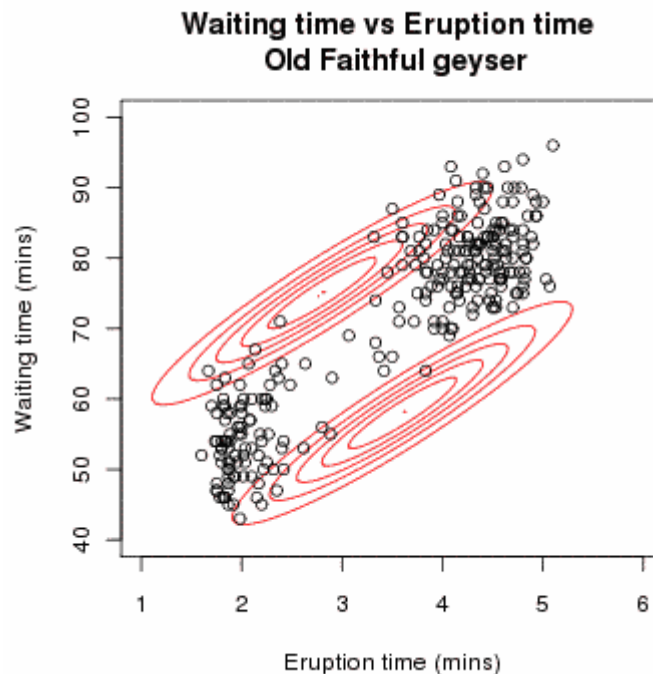
1. It uses the length of the pose vector to represent the probability that the entity represented by a capsule is present. To keep the length less than 1 requires an unprincipled non-linearity that prevents there from being any sensible objective function that is minimized by the iterative routing procedure.
2. It uses the cosine of the angle between two pose vectors to measure their agreement. Unlike the log variance of a Gaussian cluster, the cosine is not good at distinguishing between quite good agreement and very good agreement
3. It uses a vector of length  $n$  rather than a matrix with  $n$  elements to represent a pose, so its transformation matrices have  $n^2$  parameters rather than just  $n$ .

## 3.2 Model detail

### 1. Model Graph



### 2. Gaussian mixture



### 3. EM routing



**Procedure 1** Routing algorithm<sup>1</sup> returns **activation** and **pose** of the capsules in layer  $L + 1$  given the **activations** and **votes** of capsules in layer  $L$ .  $V_{ich}$  is an  $H$  dimensional vote from capsule  $i$  with activation  $a_i$  in layer  $L$  to capsule  $c$  in layer  $L + 1$ .  $\beta_a, \beta_v$  are learned discriminatively and the inverse temperature  $\lambda$  increases at each iteration with a fixed schedule.

```

1: procedure EM ROUTING( $\mathbf{a}, V$ )
2:    $\forall i, c: R_{ic} \leftarrow 1/\text{size}(L + 1)$ 
3:   for  $t$  iterations do
4:      $\forall c: M_{c:}, S_{c:}, \mathbf{a}'_c \leftarrow \text{M-STEP}(R_{c:}, \mathbf{a}, V_{c:})$ 
5:      $\forall i: R_{i:} \leftarrow \text{E-STEP}(M, S, \mathbf{a}', V_{i:})$ 
   return  $\mathbf{a}', M$ 

```

```

1: procedure M-STEP( $\mathbf{r}, \mathbf{a}, V'$ )
2:    $\forall i: \mathbf{r}'_i \leftarrow \mathbf{r}_i * \mathbf{a}_i$ 
3:    $\forall h: \mu_h \leftarrow \frac{\sum_i \mathbf{r}'_i V'_{ih}}{\sum_i \mathbf{r}'_i}$ 
4:    $\forall h: \sigma_h^2 \leftarrow \frac{\sum_i \mathbf{r}'_i (V'_{ih} - \mu_h)^2}{\sum_i \mathbf{r}'_i}$ 
5:    $\text{cost}_h \leftarrow (\beta_v + \log(\sigma_h)) \sum_i \mathbf{r}'_i$ 
6:    $\mathbf{a}' \leftarrow \text{sigmoid}(\lambda(\beta_a - \sum_h \text{cost}_h))$ 
7:   return  $\mu, \sigma, \mathbf{a}'$ 

```

▷ for one higher-level capsule

```

1: procedure E-STEP( $\mathbf{a}', S, M, V''$ )
2:    $\forall c: p_c \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi S_{ch}^2}} e^{-\sum_h^H \frac{(V''_{ch} - M_{ch})^2}{2S_{ch}^2}}$ 
3:    $\forall c: \mathbf{r}_c \leftarrow \frac{\mathbf{a}'_c p_c}{\sum_j \mathbf{a}'_j p_j}$ 
4:   return  $\mathbf{r}$ 

```

▷ for one lower-level capsule

#### 4. Experiment Result: ADVERSARIAL ROBUSTNESS

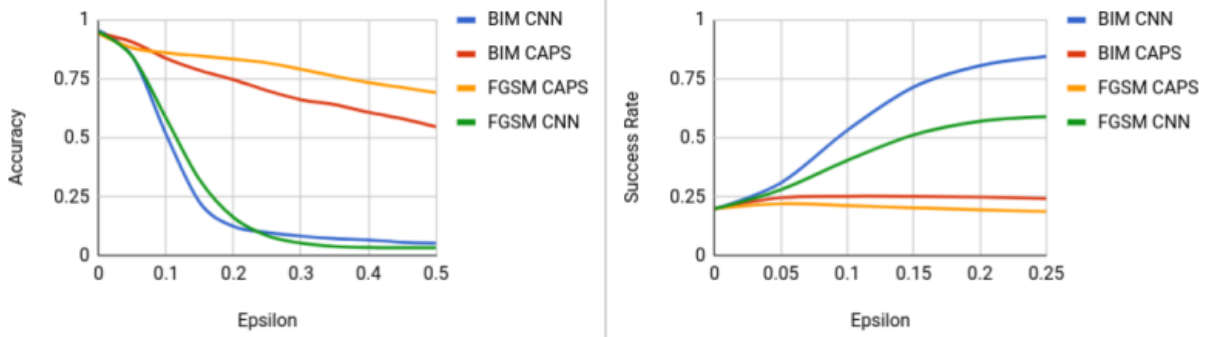


Figure 3: Accuracy against  $\epsilon$  after an adversarial attack (left) and Success Rate after a targeted adversarial attack (right). The targeted attack results were evaluated by averaging the success rate after the attack for each of the 5 possible classes.