Capsule Networks

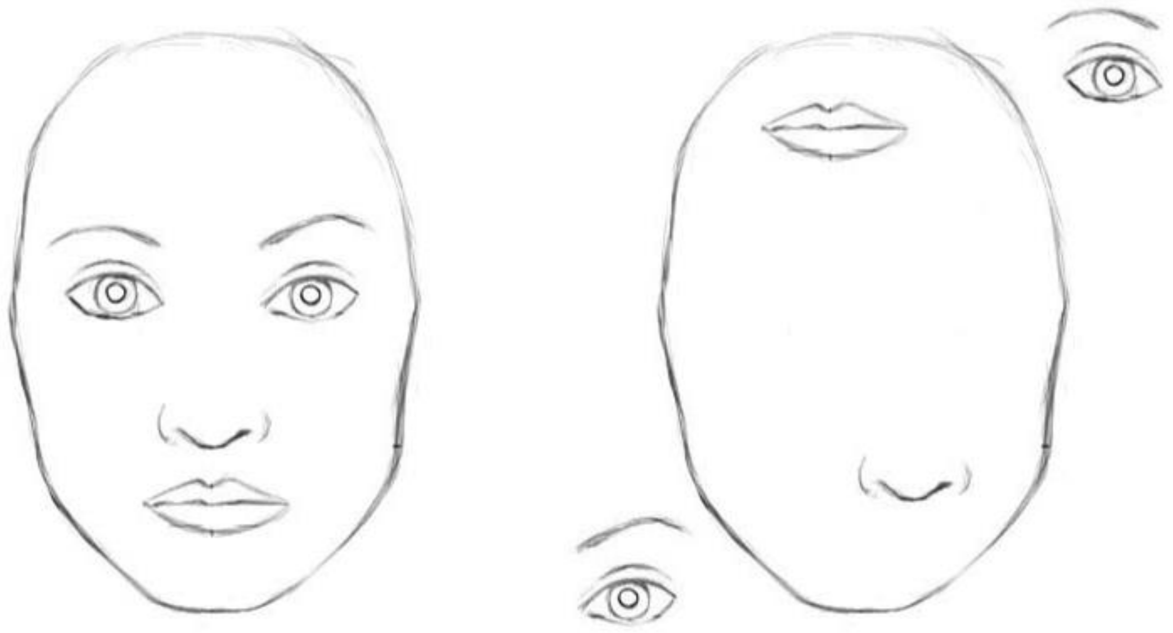
Changyou Chen
Department of Computer Science and Engineering
University at Buffalo, SUNY changyou@buffalo.edu

March 12, 2019

Reference: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

Drawback of CNNs

- CNNs fail to detect orientational and relative spatial relationships between components (objectives).



To a CNN, both pictures are similar, since they both contain similar elements

Capsule Networks

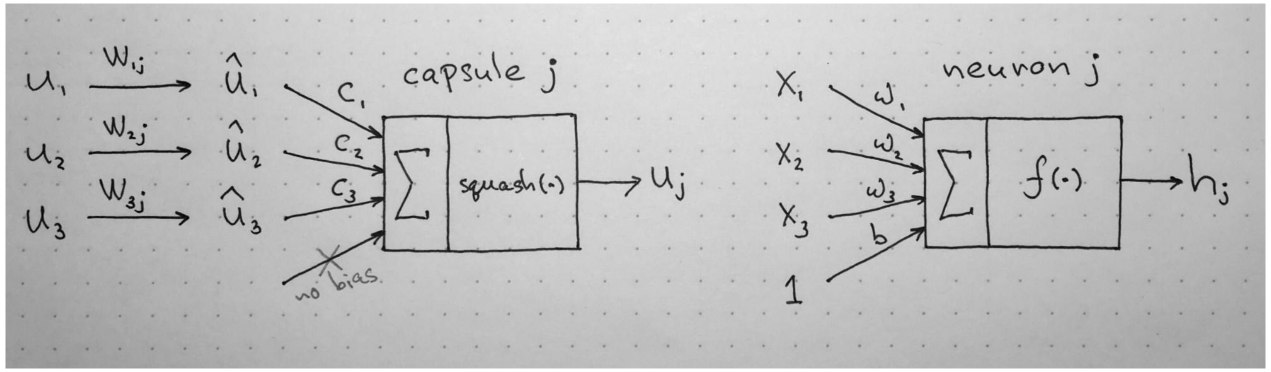
- Hinton argues that in order to correctly do classification and object recognition, it is important to preserve hierarchical pose relationships between object parts.
- Capsule outputs a vector in each neuron, encoding probability of detection of a feature as the length of the output vector.
- The state of the detected feature is encoded as the direction in which that vector points to (“instantiation parameters”).

Capsule Networks

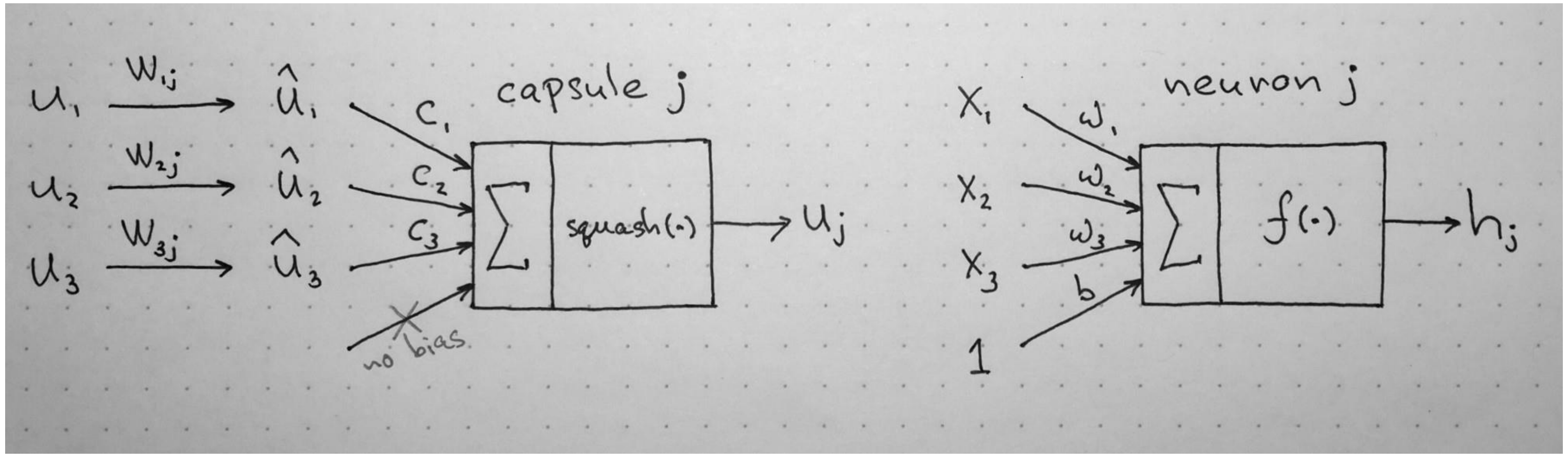
- Imagine that a capsule detects a face in the image and outputs a 3D vector of length 0.99.
- Then we start moving the face across the image. The vector will rotate in its space, representing the changing state of the detected face, but its length will remain fixed, because the capsule is still sure it has detected a face.
- This is what Hinton refers to as activities equivariance: neuronal activities will change when an object “moves over the manifold of possible appearances” in the picture. At the same time, the probabilities of detection remain constant.

How Does a Capsule Work

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\widehat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	$-$
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\widehat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\\|\mathbf{s}_j\\|^2}{1+\\|\mathbf{s}_j\\|^2}\frac{\mathbf{s}_j}{\\|\mathbf{s}_j\\|}$	$h_j = f(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)

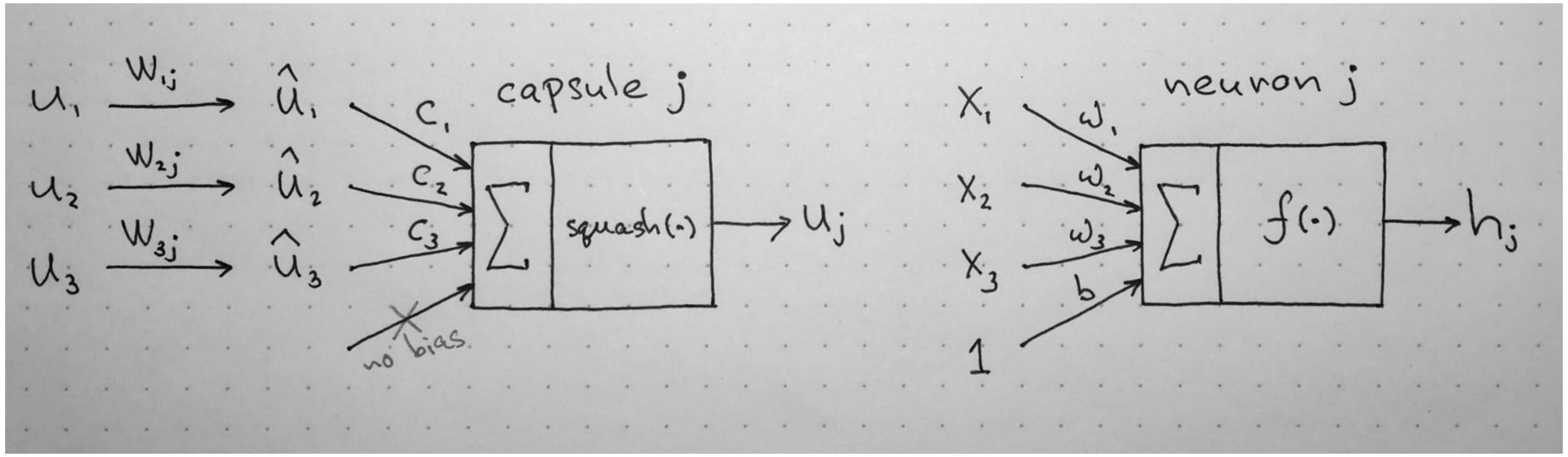


Matrix Multiplication of Input Vectors



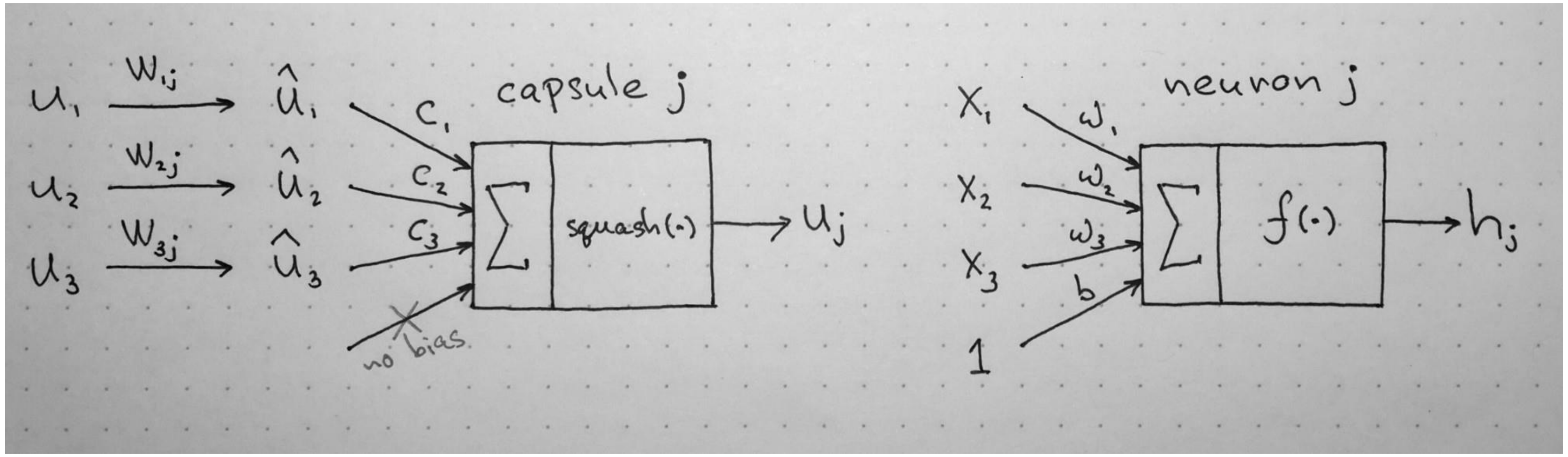
- Input vectors u_1, u_2 and u_3 come from 3 other capsules in the layer below.
- Lengths encode probabilities that lower-level capsules detected their corresponding objects.
- Directions encode some internal state of the detected objects.
  - e.g., lower level capsules detect eyes, mouth and nose respectively; and output capsule detects face.

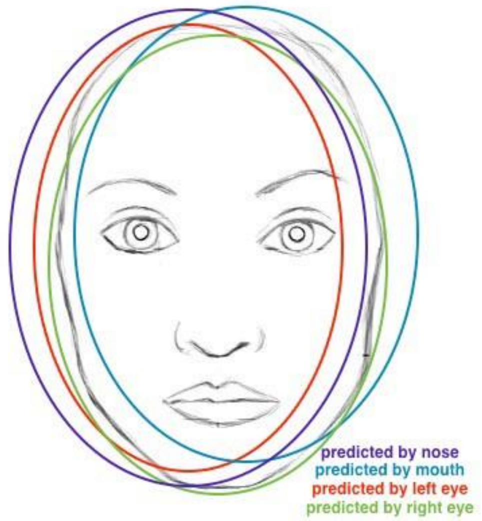
Matrix Multiplication of Input Vectors



- These vectors are multiplied by corresponding weight matrices W that encode important spatial and other relationships between lower level features (eyes, mouth and nose) and higher level feature (face).
  - e.g., W_{2j} may encode relationship between nose and face: face is centered around its nose, its size is 10 times the size of the nose and its orientation in space corresponds to orientation of the nose

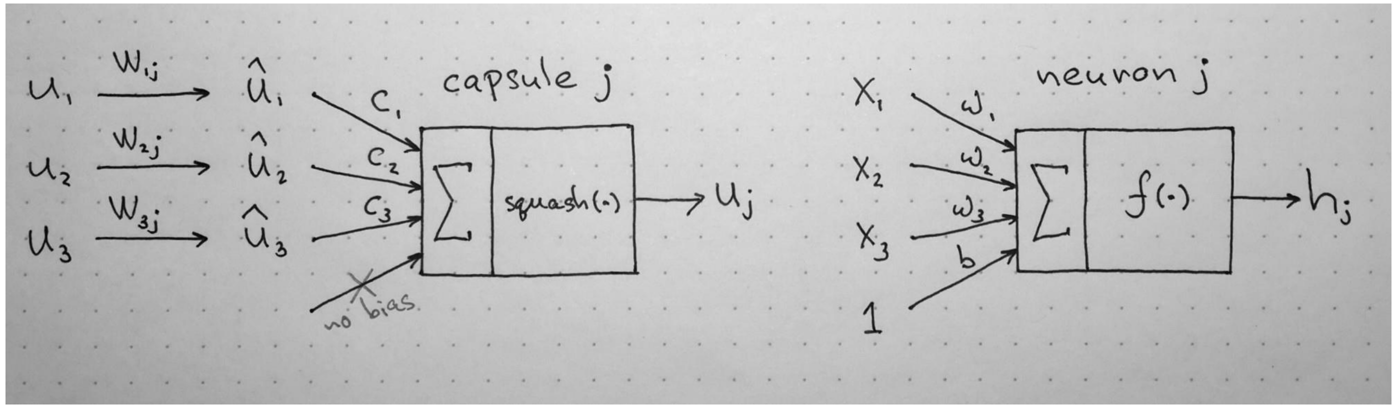
Matrix Multiplication of Input Vectors



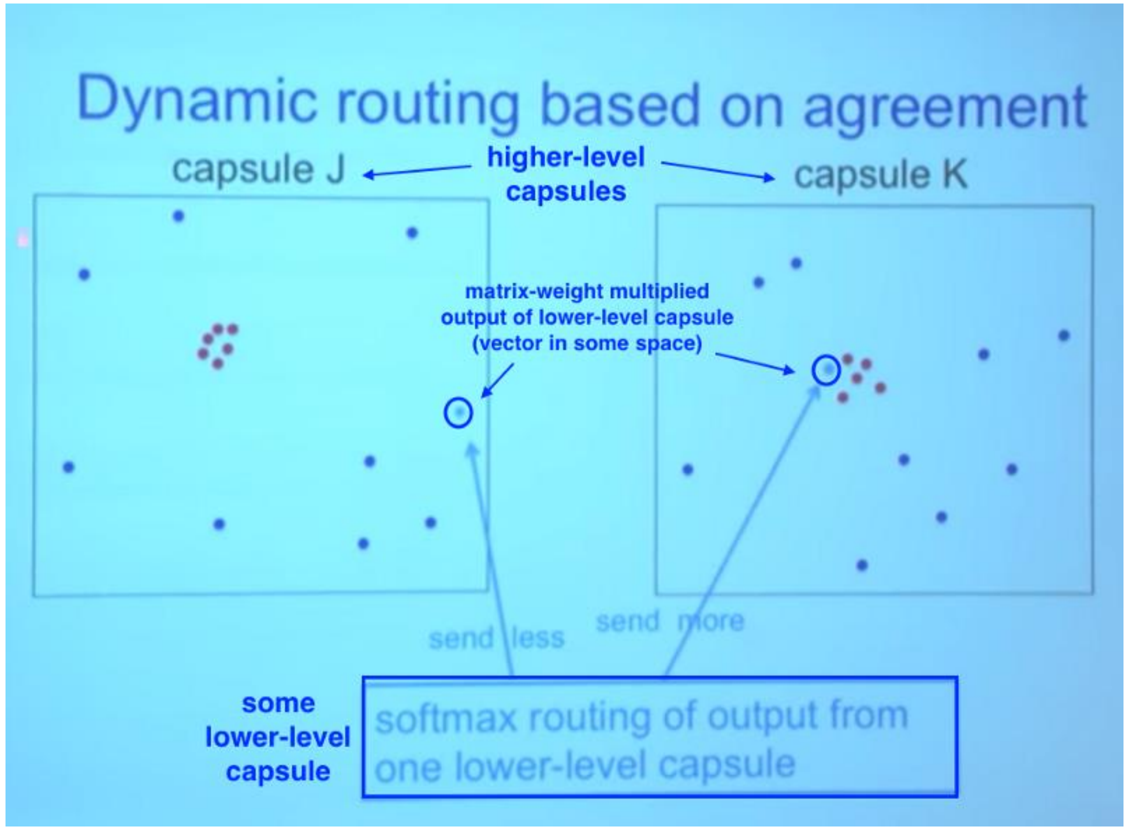


- After multiplication, what we get is the predicted position of the higher level feature.
  - e.g., $\hat{u}_1$ represents where the face should be according to the detected position of the eyes; $\hat{u}_2$represents where the face should be according to the detected position of the mouth; $\hat{u}_3$represents where the face should be according to the detected position of the nose.
- Intuition: if these 3 predictions of lower level features point at the same position and state of the face, then it must be a face there.

Scalar Weighting of Input Vectors



- Weights c_i decided by "dynamic routing", different from BP.

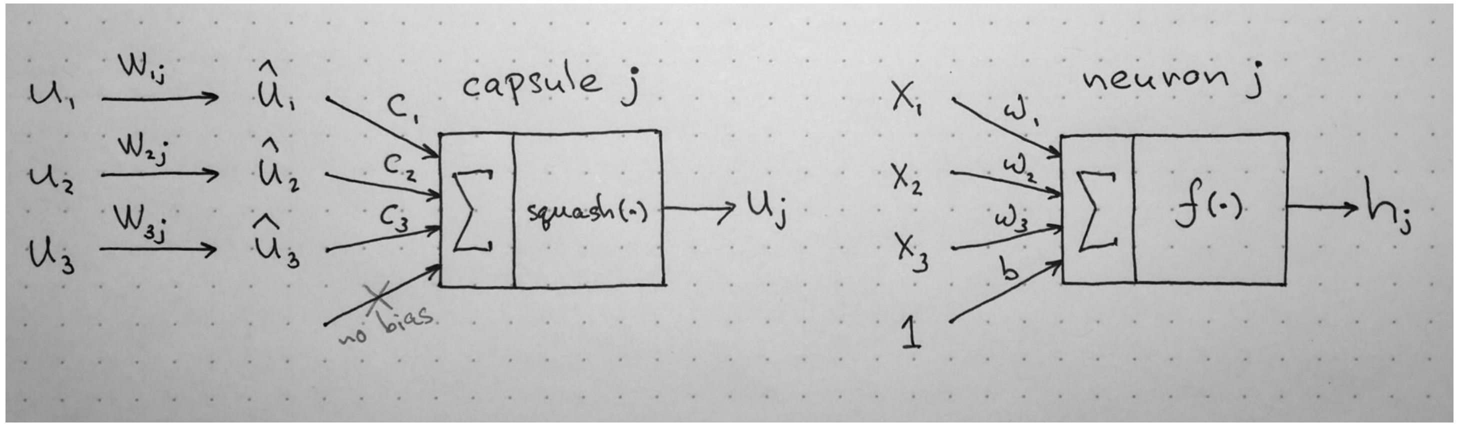


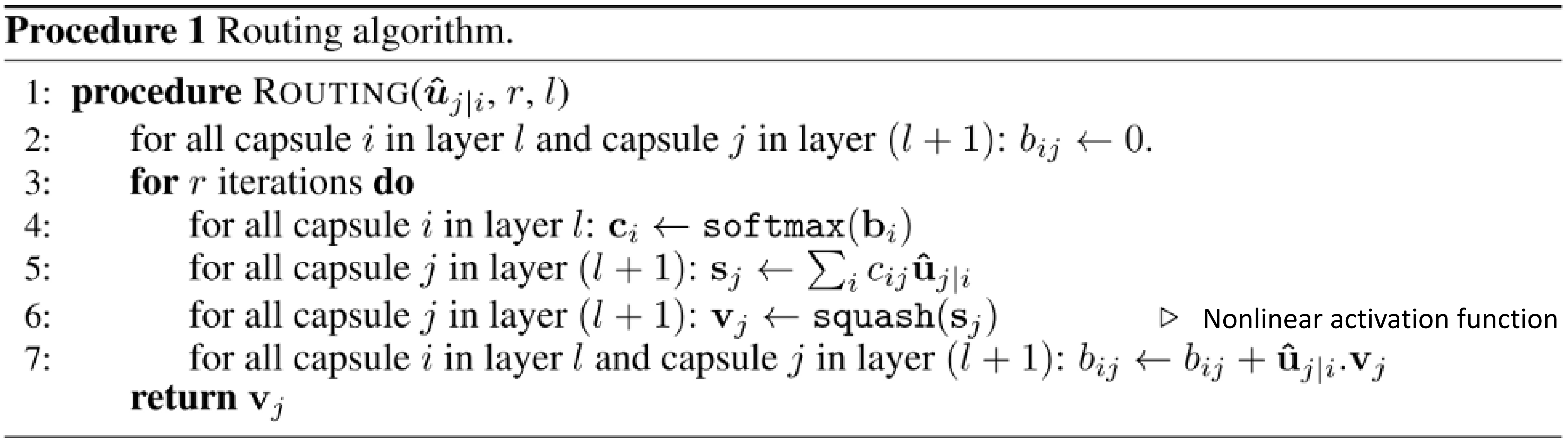
Dynamic routing based on agreement
capsule J
higher-level capsules
capsule K
matrix-weight multiplied output of lower-level capsule (vector in some space)
send less
send more
some lower-level capsule
softmax routing of output from one lower-level capsule

Lower level capsule will send its input to the higher level capsule that "agrees" with its input, by adjusting its weights c_i.

Dynamic Routing Between Capsules

- Lower level capsule will send its input to the higher level capsule that "agrees" with its input.
- Adjusting $c_i's$ such that $\sum_i c_i = 1$

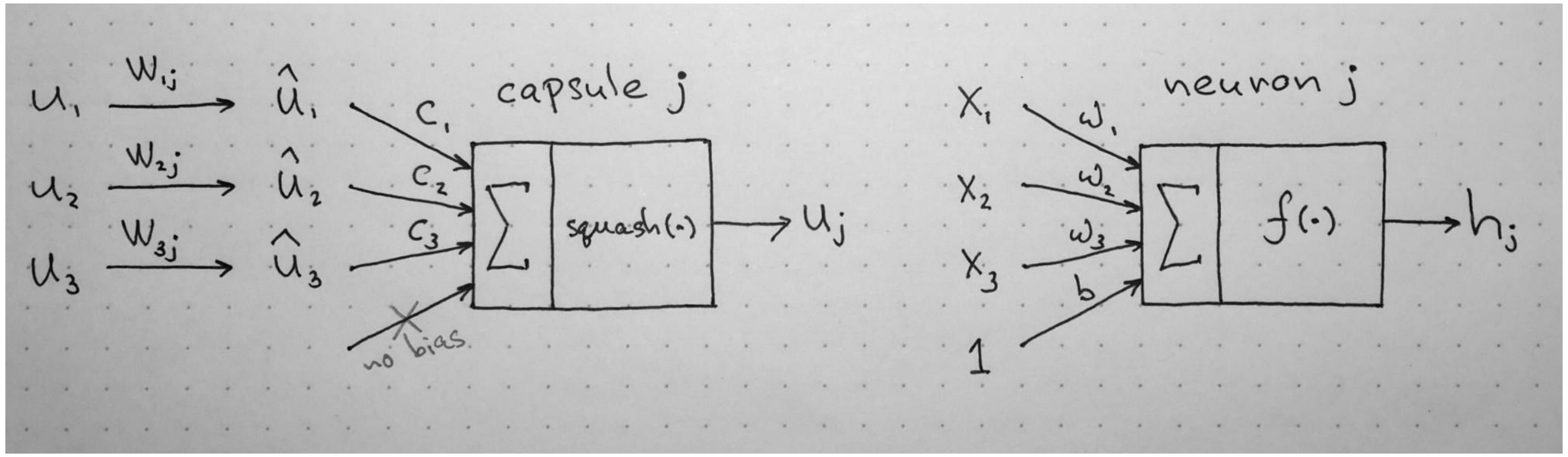


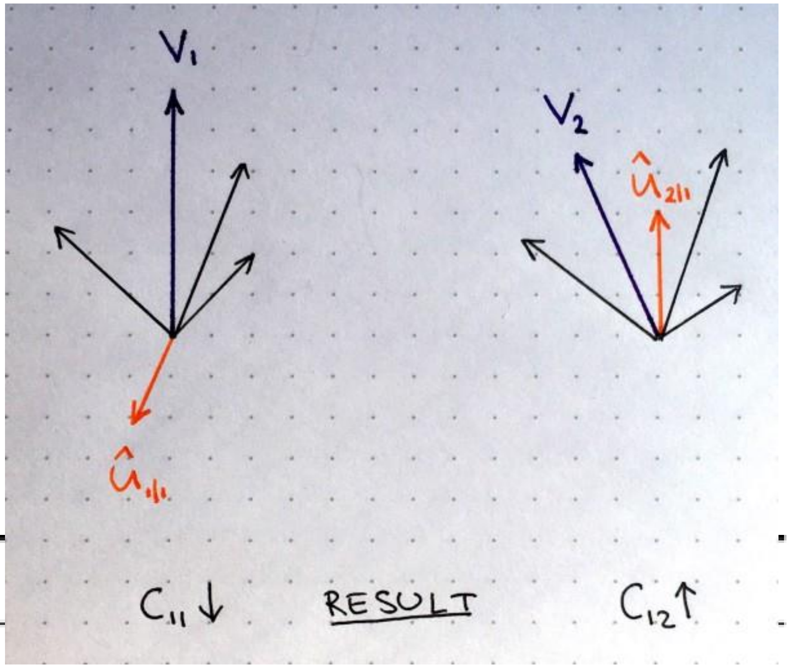


Procedure 1 Routing algorithm.

```
1: procedure 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 $l$: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$
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 $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$      ▷ Nonlinear activation function
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
    return $\mathbf{v}_j$
```

Dynamic Routing Between Capsules: Example

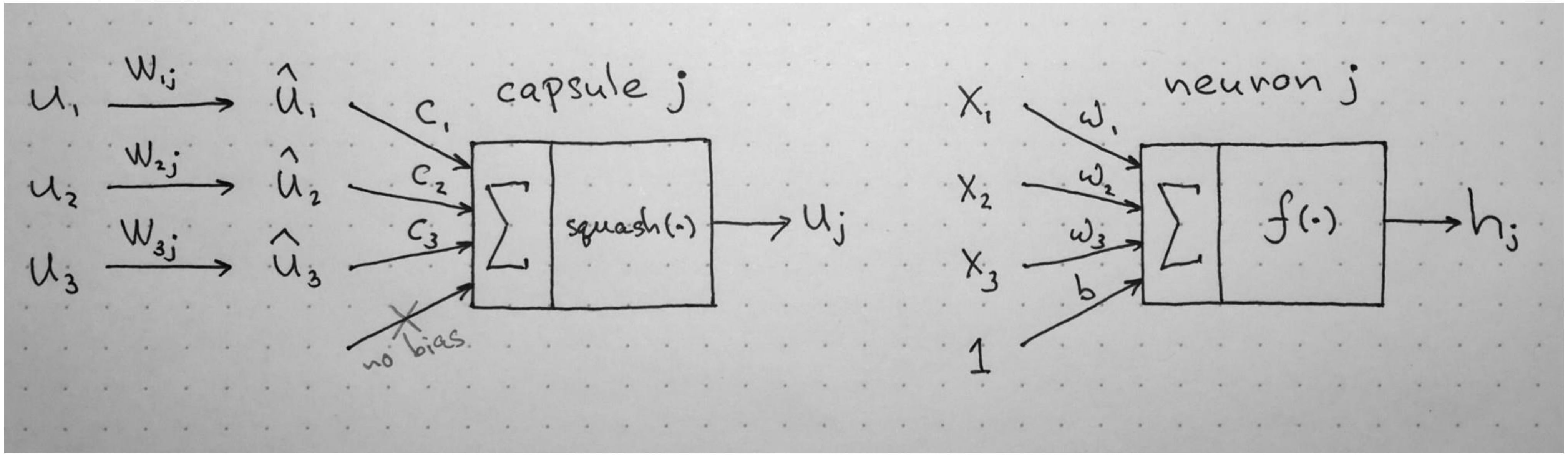




Procedure 1 Routing algorithm.

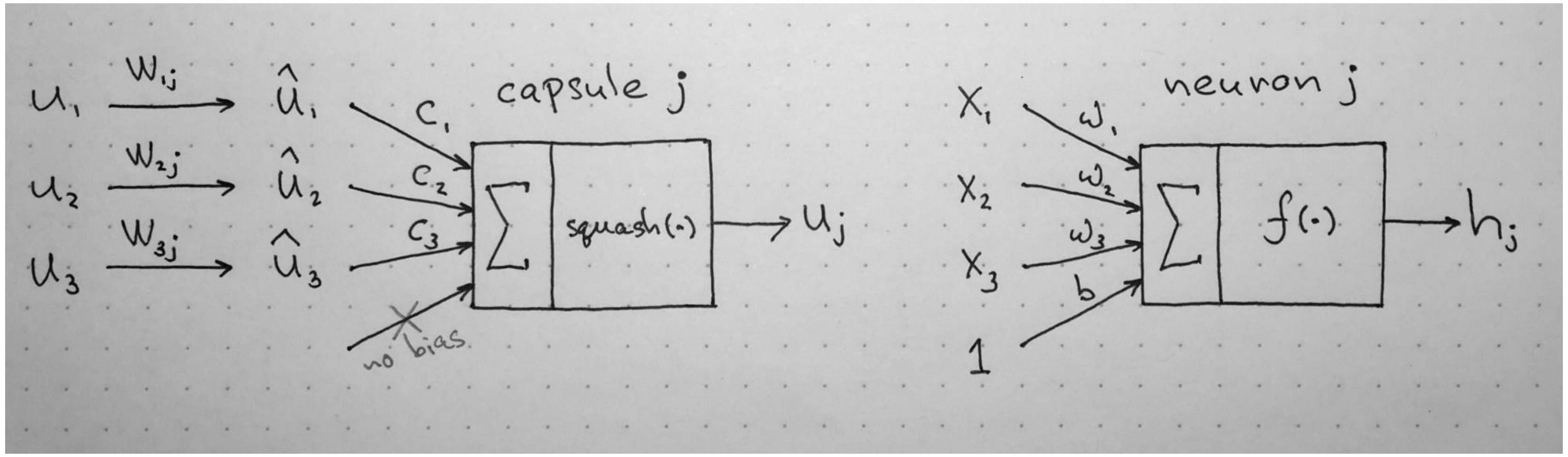
```
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
5:         for 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 $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$      ▷ Nonlinear activation function
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
    return $\mathbf{v}_j$
```

Sum of Weighted Input Vectors

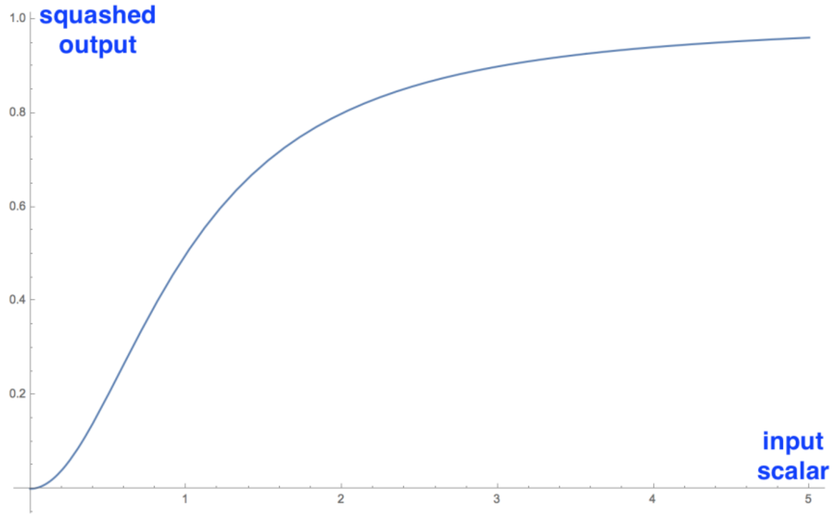


- Similar to the standard DNN.

"Squash": Novel Vector-to-Vector Nonlinearity

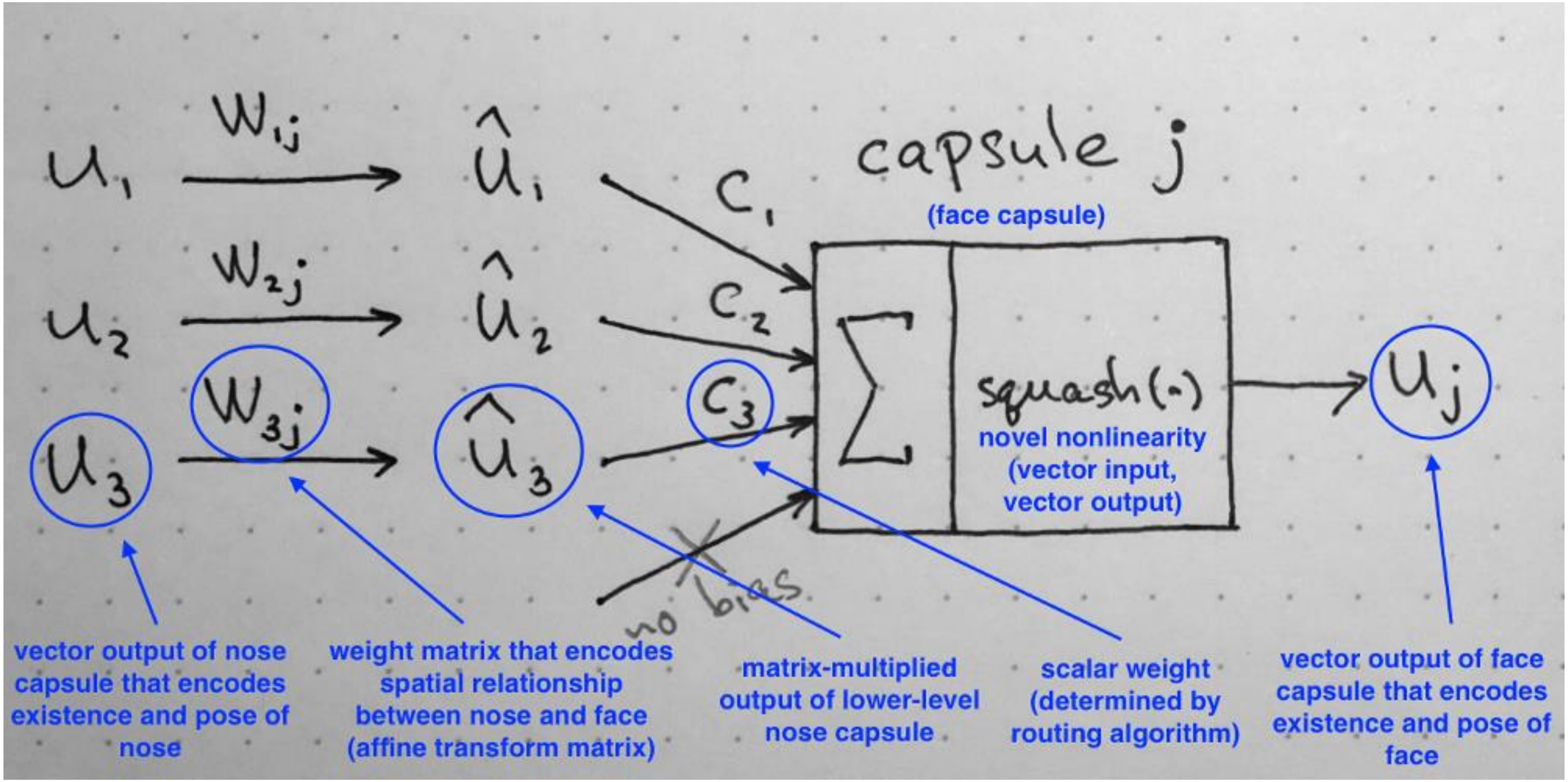


$$\mathbf{v}_j = \underbrace{\frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}}_{\text{additional "squashing"}} \underbrace{\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}}_{\text{unit scaling}}$$



Squashing nonlinearity scales input vector without changing its direction.

Summary

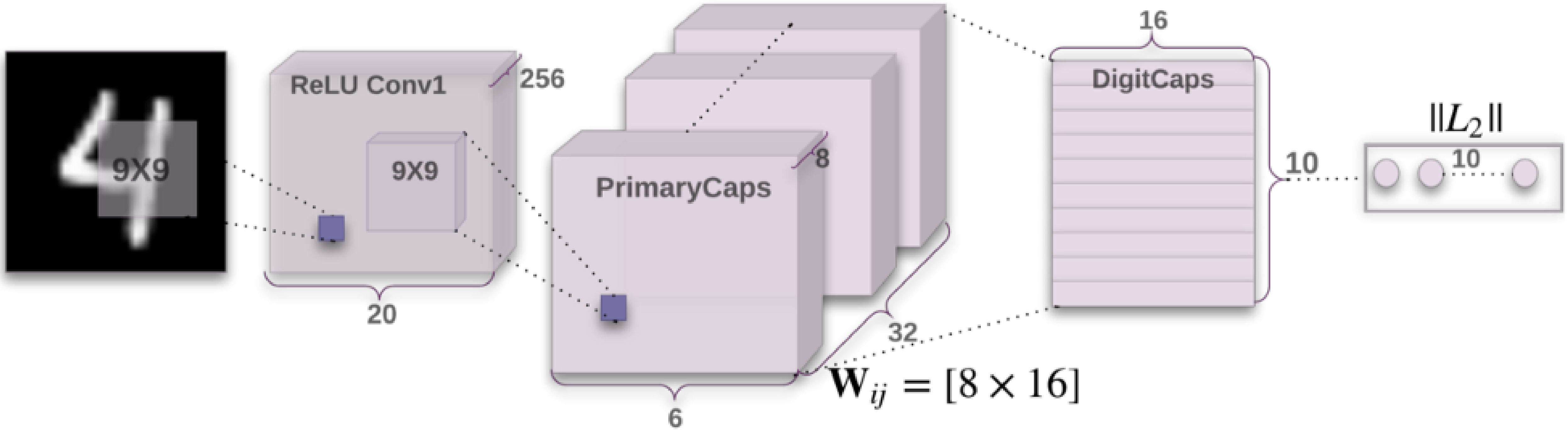


u_1
u_2
u_3
W_{1j}
W_{2j}
W_{3j}
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
c_1
c_2
c_3
capsule j
(face capsule)
Σ
squash(·)
novel nonlinearity (vector input, vector output)
u_j
no bias
vector output of nose capsule that encodes existence and pose of nose
weight matrix that encodes spatial relationship between nose and face (affine transform matrix)
matrix-multiplied output of lower-level nose capsule
scalar weight (determined by routing algorithm)
vector output of face capsule that encodes existence and pose of face

CapsNet Architecture

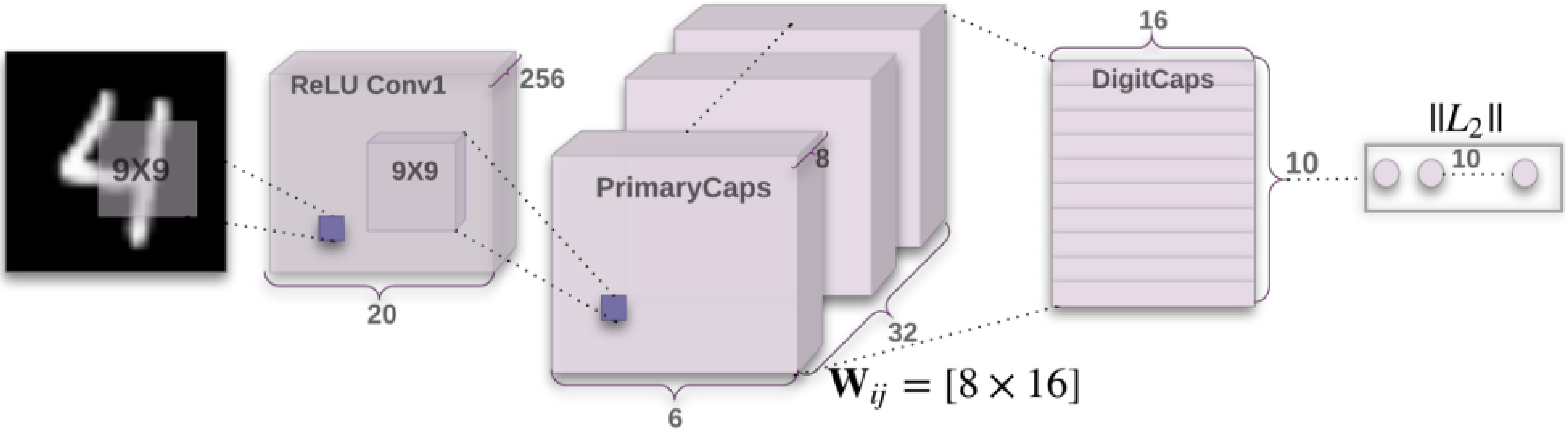
- The CapsNet has 2 parts: encoder and decoder. The first 3 layers are encoder, and the second 3 are decoder:
  - Layer 1. Convolutional layer
  - Layer 2. PrimaryCaps layer
  - Layer 3. DigitCaps layer
  - Layer 4. Fully connected #1
  - Layer 5. Fully connected #2
  - Layer 6. Fully connected #3

CapsNet Architecture: Encoder



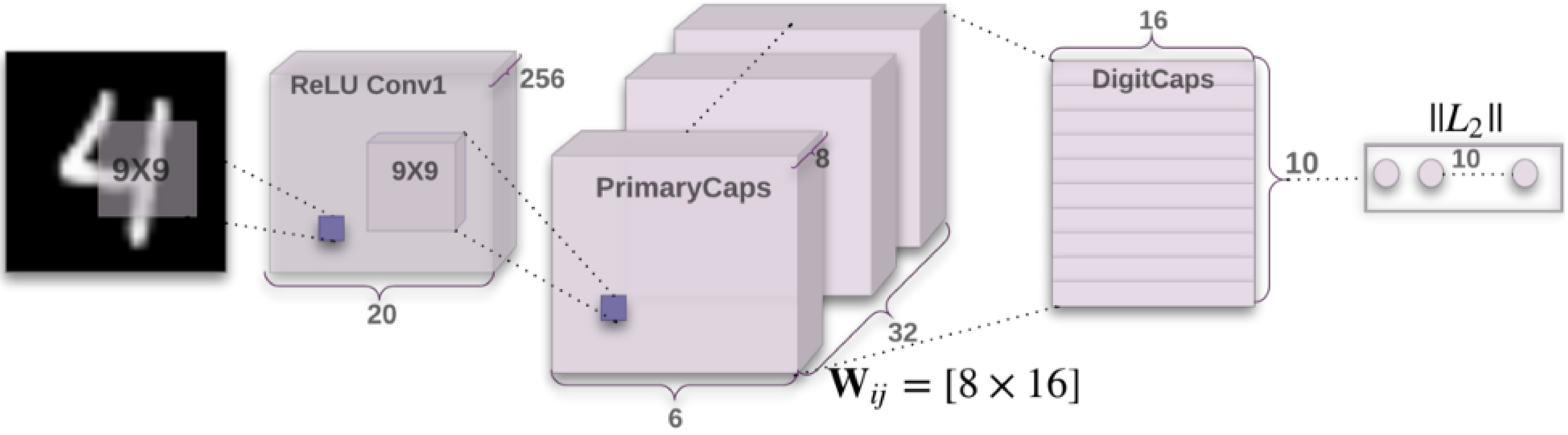
- Convolutional layer’s job is to detect basic features in the 2D image.
  - Input: 28x28 image (one color channel)
  - Output: 20x20x256 tensor
  - Output: 20x20x256 tensor.

CapsNet Architecture: Encoder



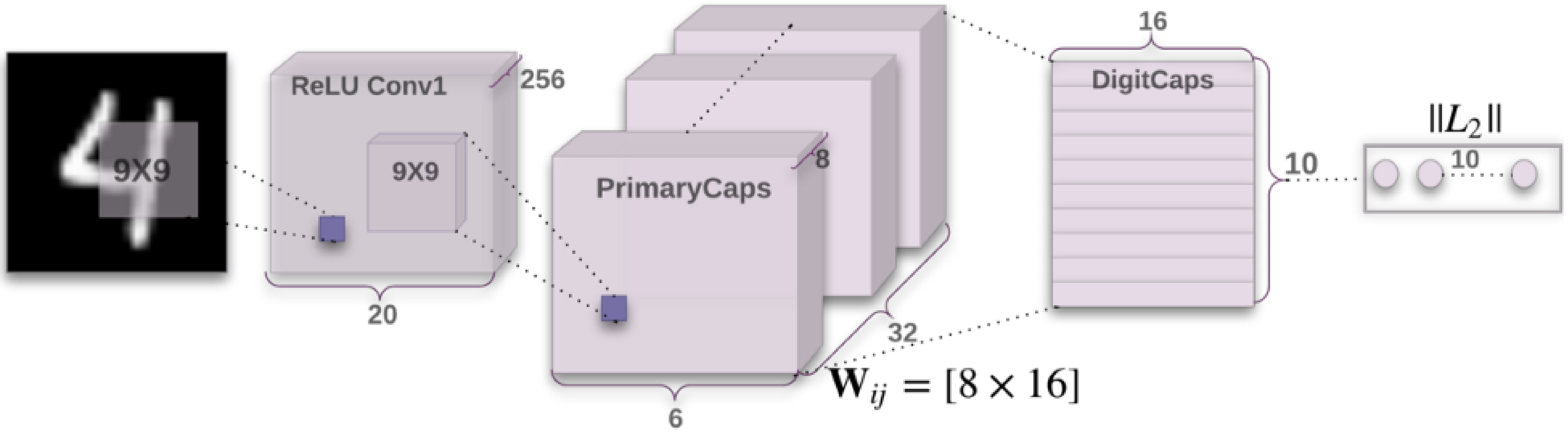
- PrimaryCaps layer has 32 primary capsules whose job is to take basic features detected by the convolutional layer and produce combinations of the features.
  - Input: 20x20x256 tensor
  - Output: 6x6x8x32 tensor
  - Number of parameters: 5308672

CapsNet Architecture: Encoder



- DigitCaps layer has 10 digit capsules, one for each digit. Each capsule takes as input a 6x6x8x32 tensor, and outputs 10 16-dimensional vectors, each for one class.
  - Input: 6x6x8x32 tensor
  - Output: 16x10 tensor
  - Number of parameters: 1497600

CapsNet Architecture: Loss Function



CapsNet Loss Function

calculated for correct DigitCap

calculated for incorrect DigitCaps

loss term for one DigitCap

L2 norm

L2 norm

$$L_c = \underbrace{T_c \max(0, m^+ - ||\mathbf{v}_c||)^2}_{\text{calculated for correct DigitCap}} + \underbrace{\lambda (1 - T_c) \max(0, ||\mathbf{v}_c|| - m^-)^2}_{\text{calculated for incorrect DigitCaps}}$$

1 when correct DigitCap, 0 when incorrect

zero loss when correct prediction with probability greater than 0.9, non-zero otherwise

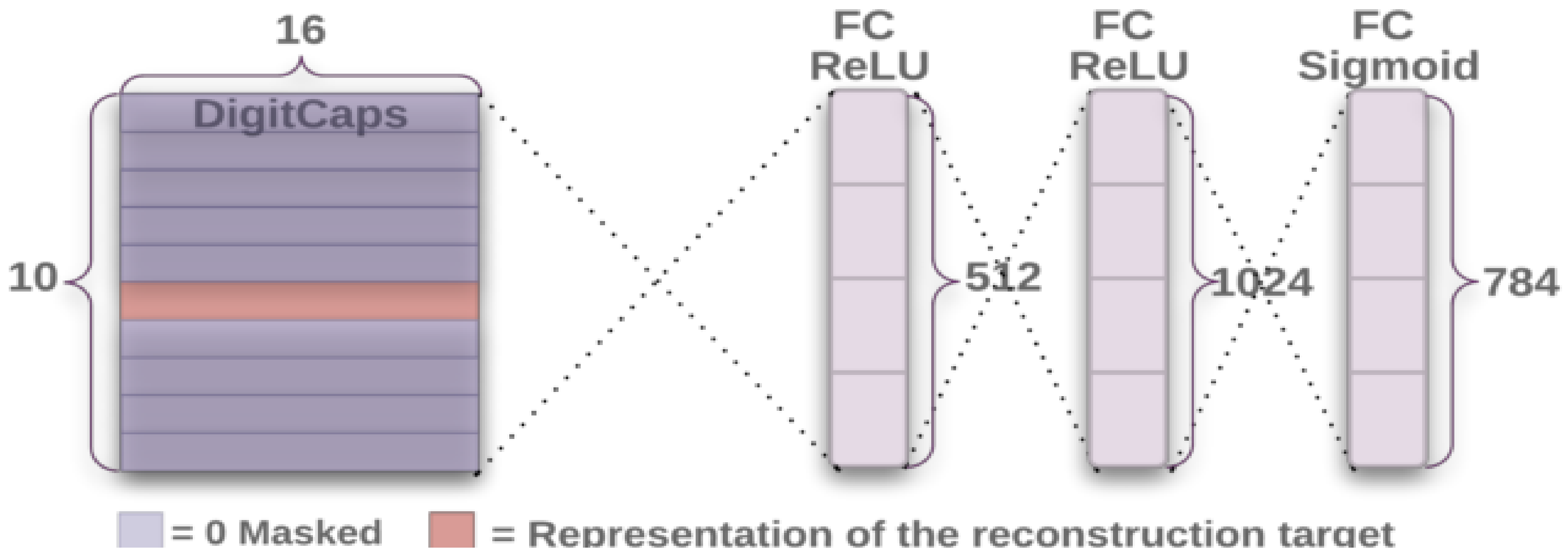
0.5 constant used for numerical stability

1 when incorrect DigitCap, 0 when correct

zero loss when incorrect prediction with probability less than 0.1, non-zero otherwise

Note: correct DigitCap is one that matches training label, for each training example there will be 1 correct and 9 incorrect DigitCaps

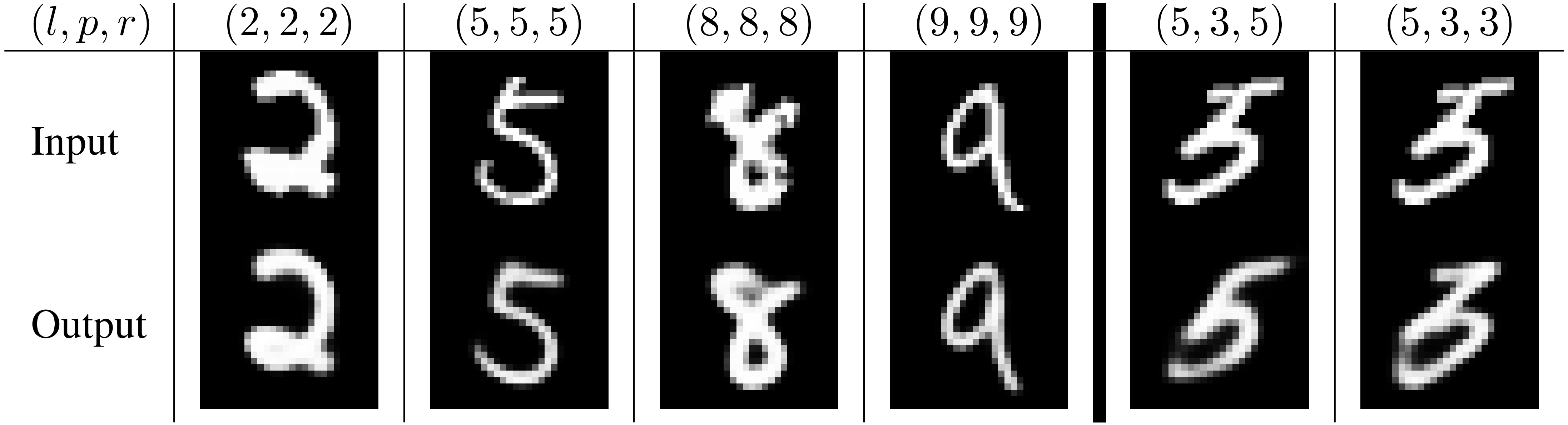
CapsNet Architecture: Decoder



- Takes a 16-dimensional vector from the correct DigitCap and learns to decode it into an image of a digit.
  - only uses the correct DigitCap vector during training and ignores the incorrect ones.
- Decoder is used as a regularizer.
  - with the loss function being Euclidean distance between the reconstructed image and the input image.

Experiments

Figure 3: Sample MNIST test reconstructions of a CapsNet with 3 routing iterations. (l, p, r) represents the label, the prediction and the reconstruction target respectively. The two rightmost columns show two reconstructions of a failure example and it explains how the model confuses a 5 and a 3 in this image. The other columns are from correct classifications and shows that model preserves many of the details while smoothing the noise.



Experiments

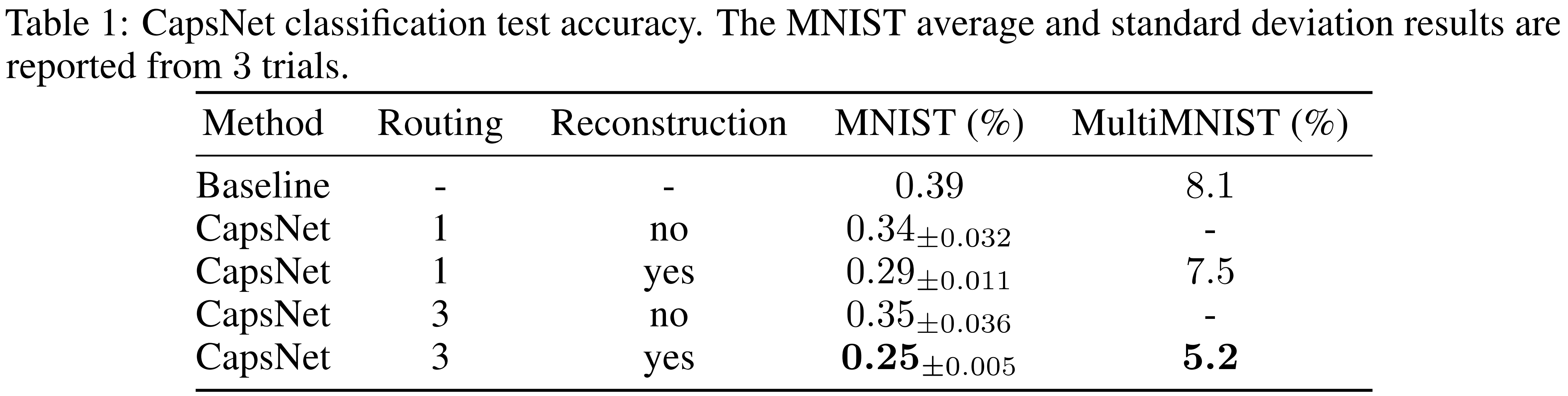


Table 1: CapsNet classification test accuracy. The MNIST average and standard deviation results are reported from 3 trials.

Method	Routing	Reconstruction	MNIST (%)	MultiMNIST (%)
Baseline	-	-	0.39	8.1
CapsNet	1	no	$0.34_{\pm 0.032}$	-
CapsNet	1	yes	$0.29_{\pm 0.011}$	7.5
CapsNet	3	no	$0.35_{\pm 0.036}$	-
CapsNet	3	yes	$\mathbf{0.25}_{\pm 0.005}$	$\mathbf{5.2}$