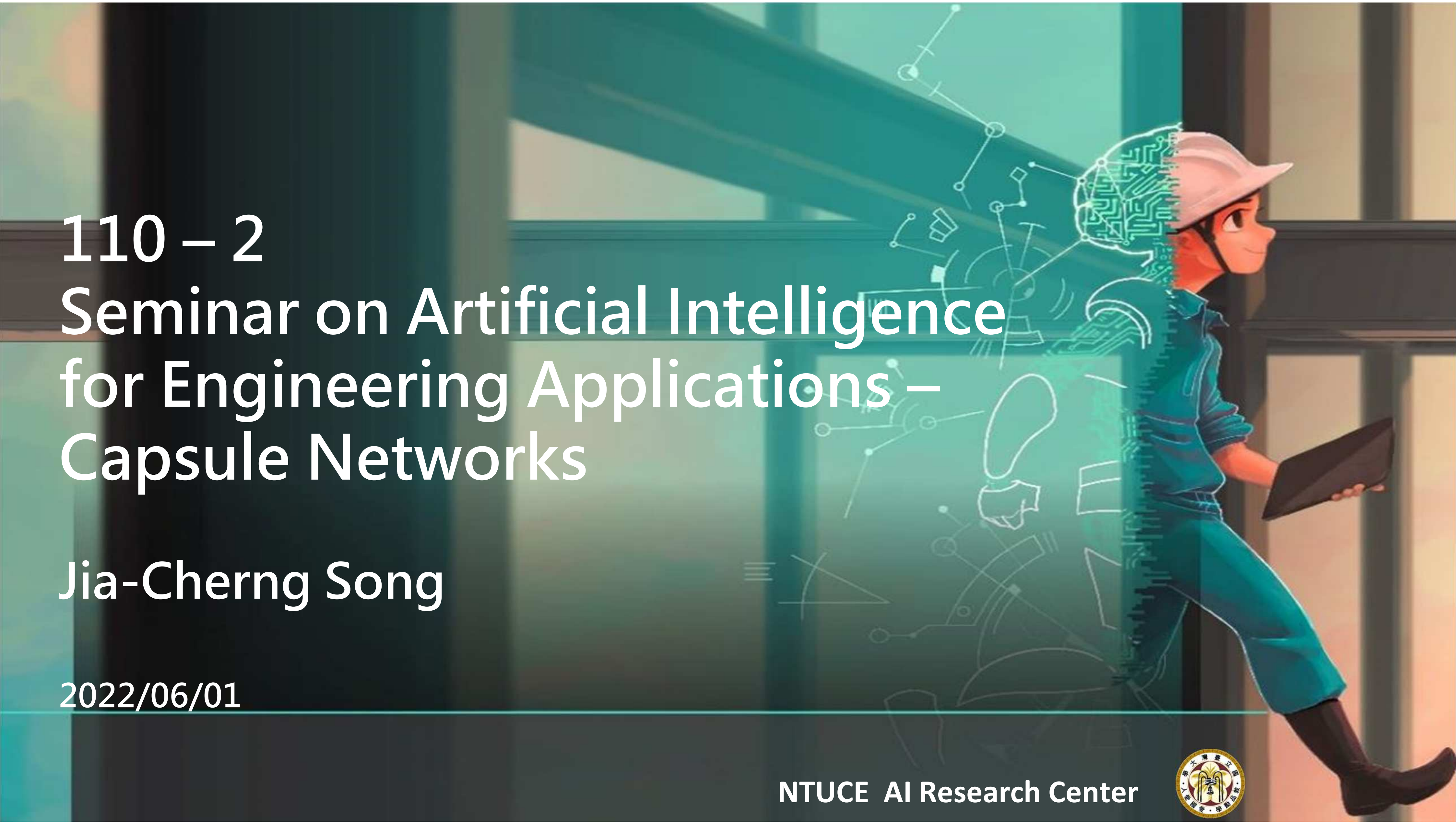


110 – 2 Seminar on Artificial Intelligence for Engineering Applications – Capsule Networks

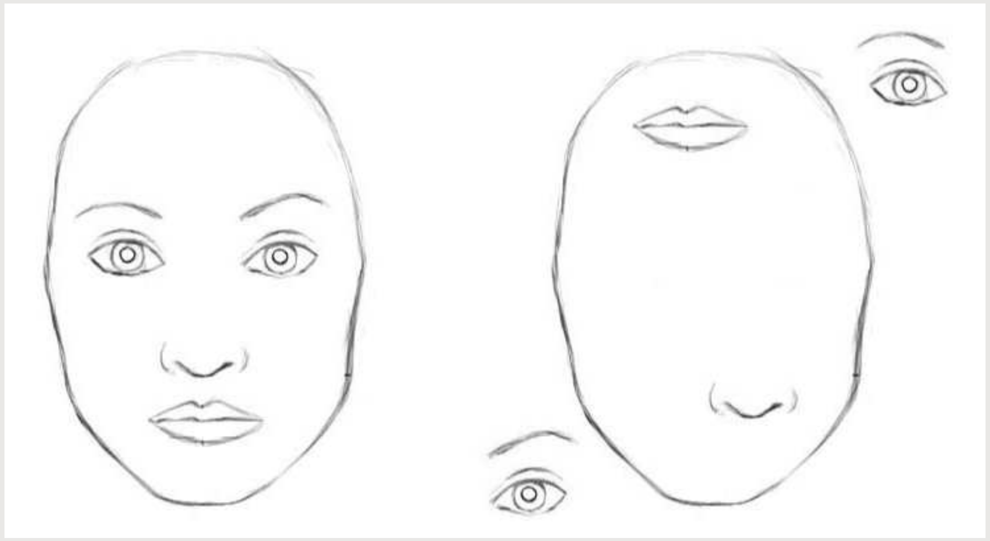
Jia-Cherng Song

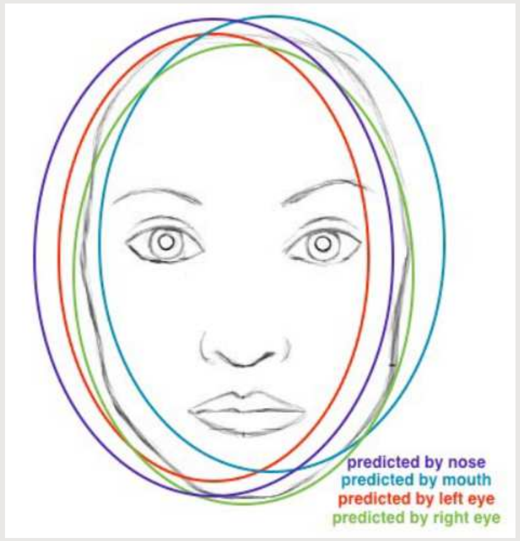
2022/06/01

NTUCE AI Research Center

Outline

Introduction





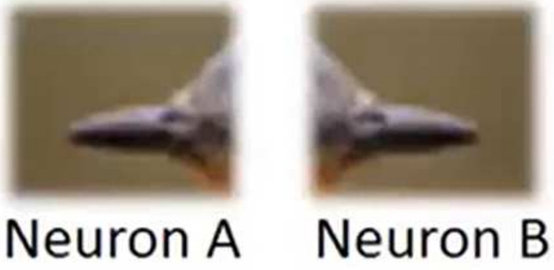
Credit: Pechyonkin, M. (2017)

“The pooling operation used in convolutional neural networks is a big mistake, and the fact that it works so well is a disaster.”

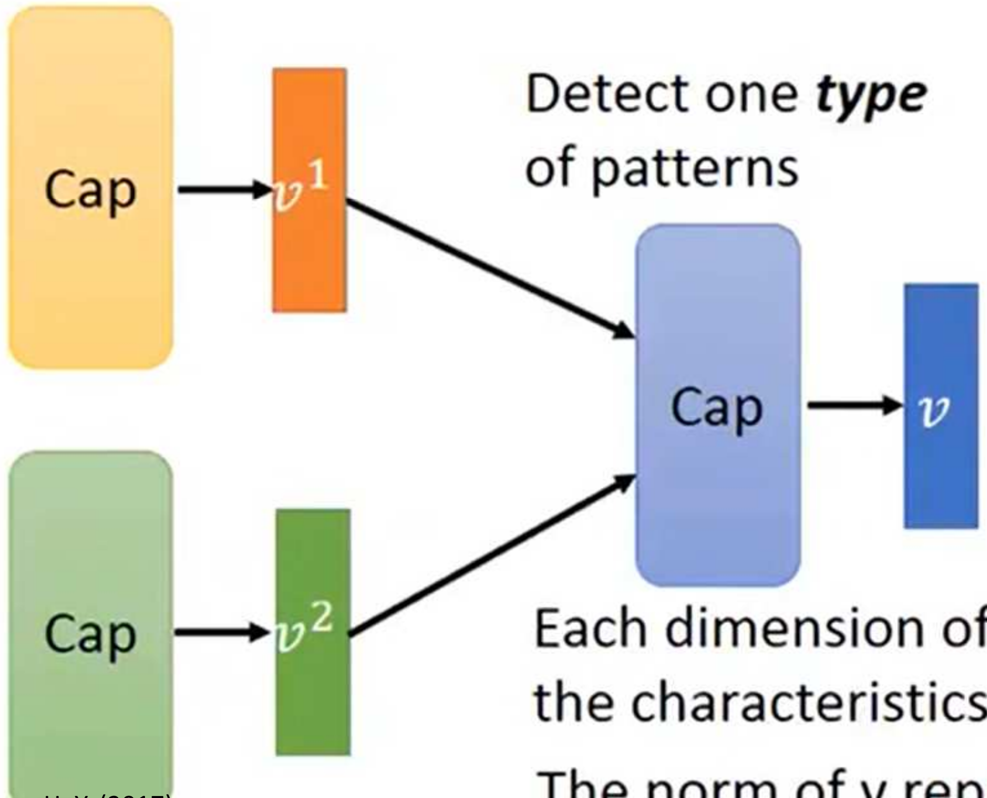
- Geoffrey Hinton

Capsule

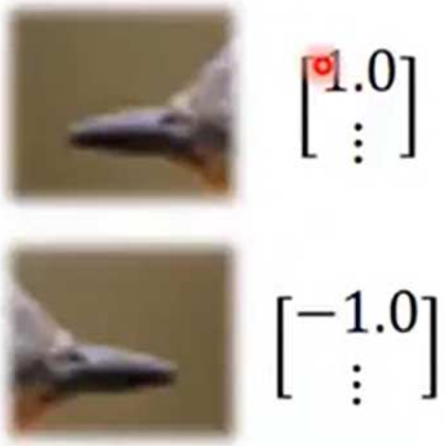
A neuron detects a specific pattern.



- Neuron: output a value, Capsule: output a vector



Detect one ***type*** of patterns

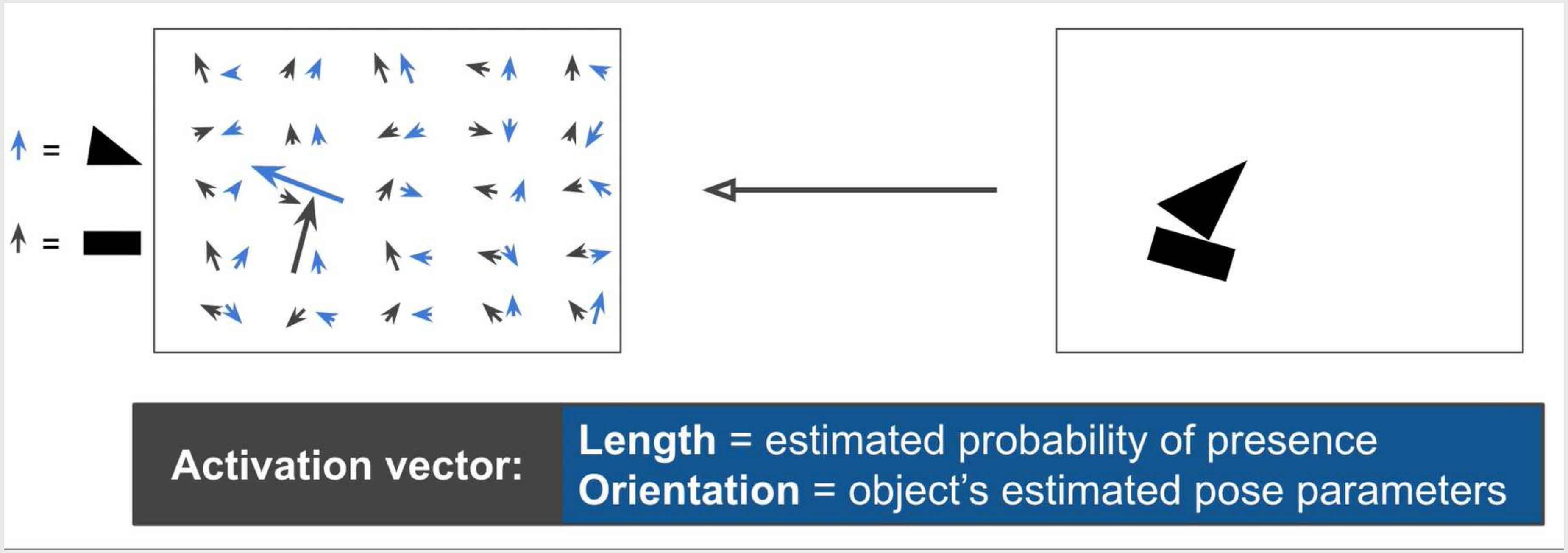

$\begin{bmatrix} 1.0 \\ \vdots \end{bmatrix}$
$\begin{bmatrix} -1.0 \\ \vdots \end{bmatrix}$

Each dimension of v represents the characteristics of patterns.

The norm of v represents the existence

Credit: Lee, H. Y. (2017)

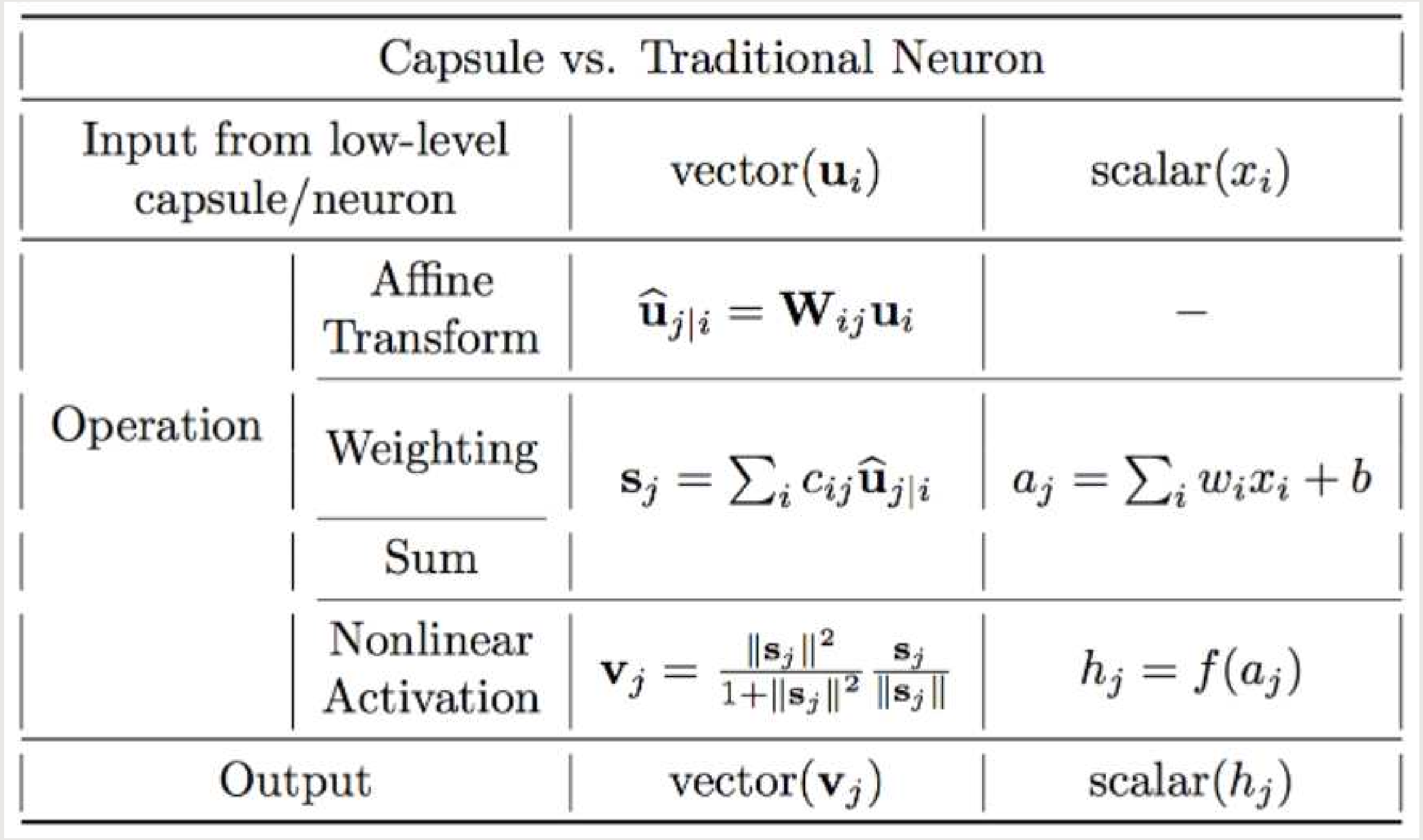
What are capsules?



Activation vector: **Length** = estimated probability of presence
Orientation = object's estimated pose parameters

Credit: Géron, A. (2017)

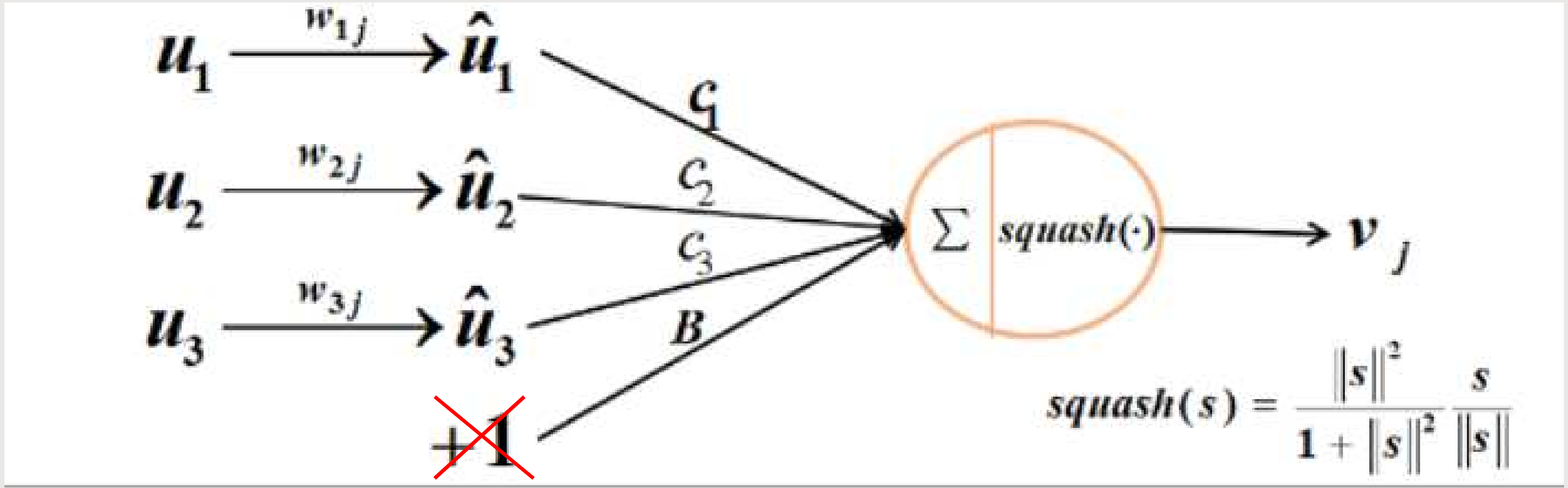
What are capsules?

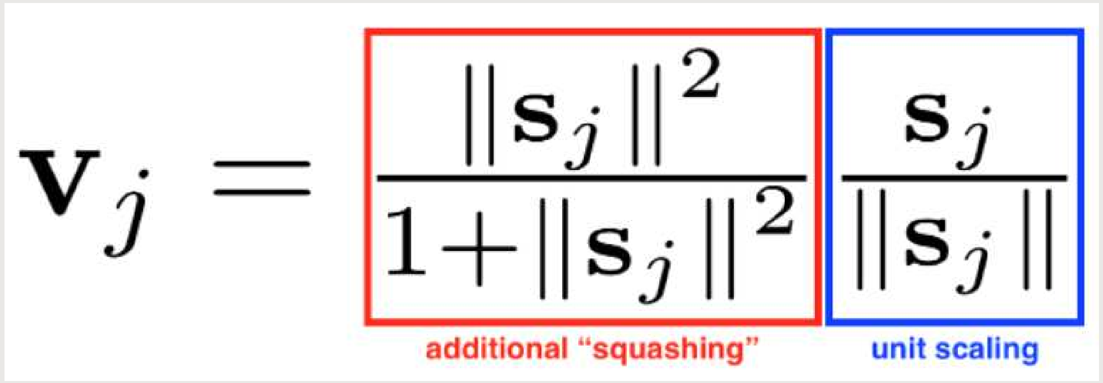


Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	$-$
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear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

Credit: Pechyonkin, M. (2017)

What are capsules?





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

If ||s|| ↑, squashing part ≊ 1
If ||s|| ↓, squashing part ≊ 0

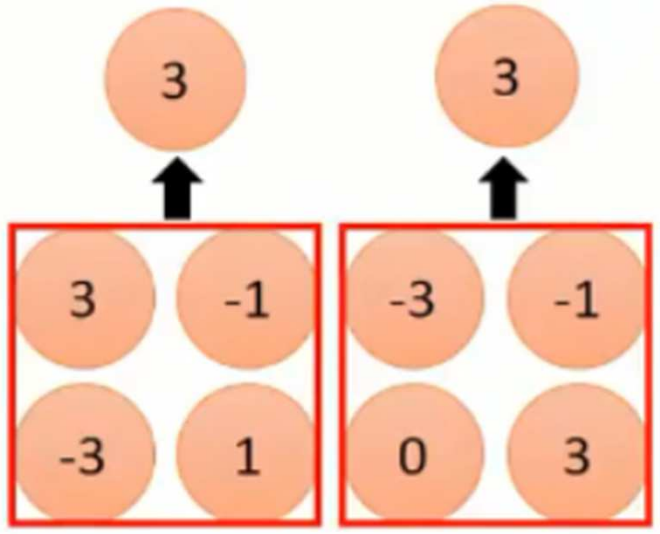
Credit: Pechyonkin, M. (2017)

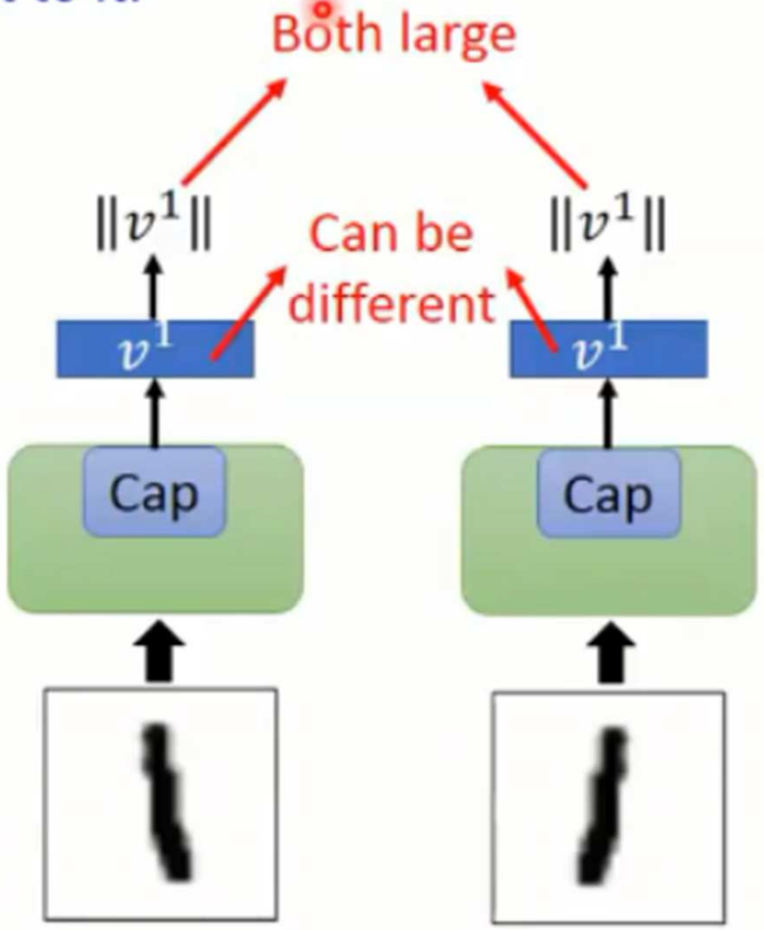
Discussion

I know the difference, but I do not react to it.

- Invariance v.s. Equivariance

I don't know the difference.





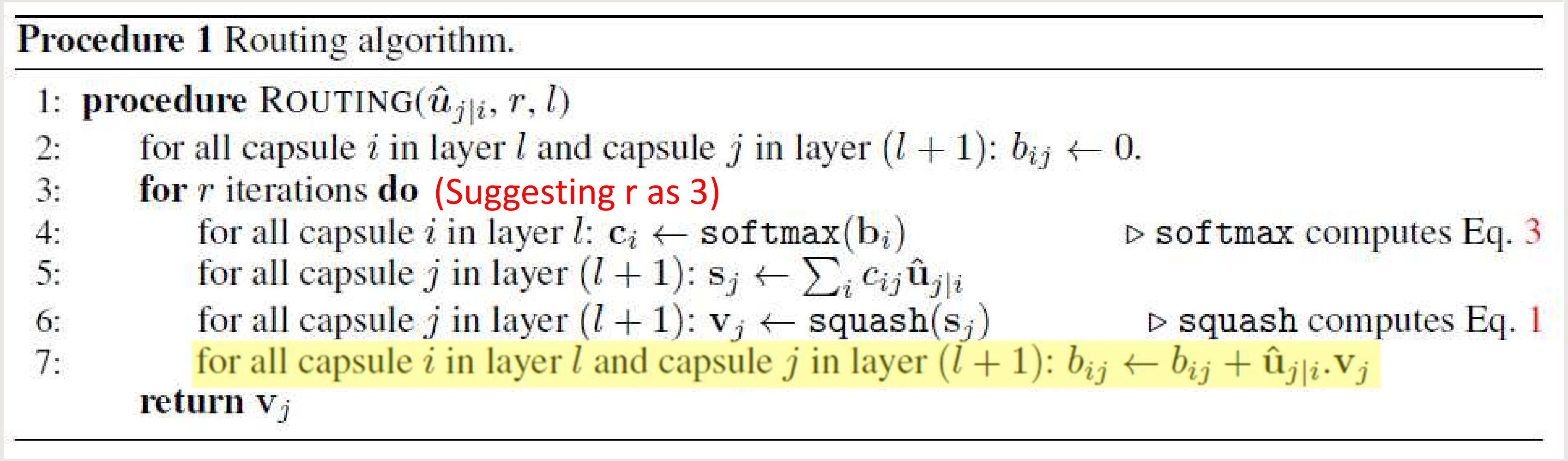
Max pooling has invariance, but not equivariance.

Capsule has both invariance and equivariance.

Created with EverCam

Credit: Lee, H. Y. (2017)

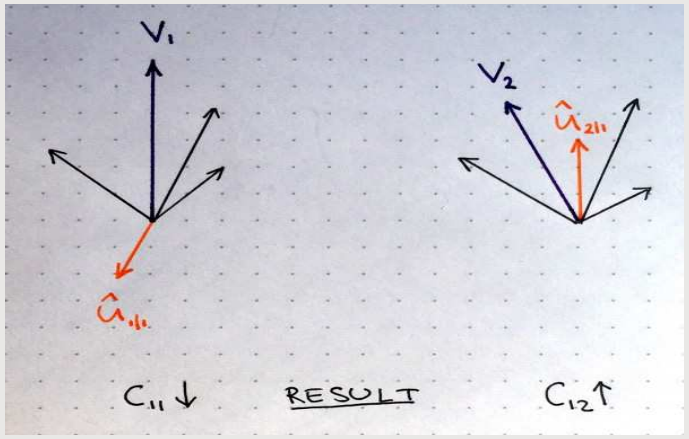
Dynamic routing



Procedure 1 Routing algorithm.

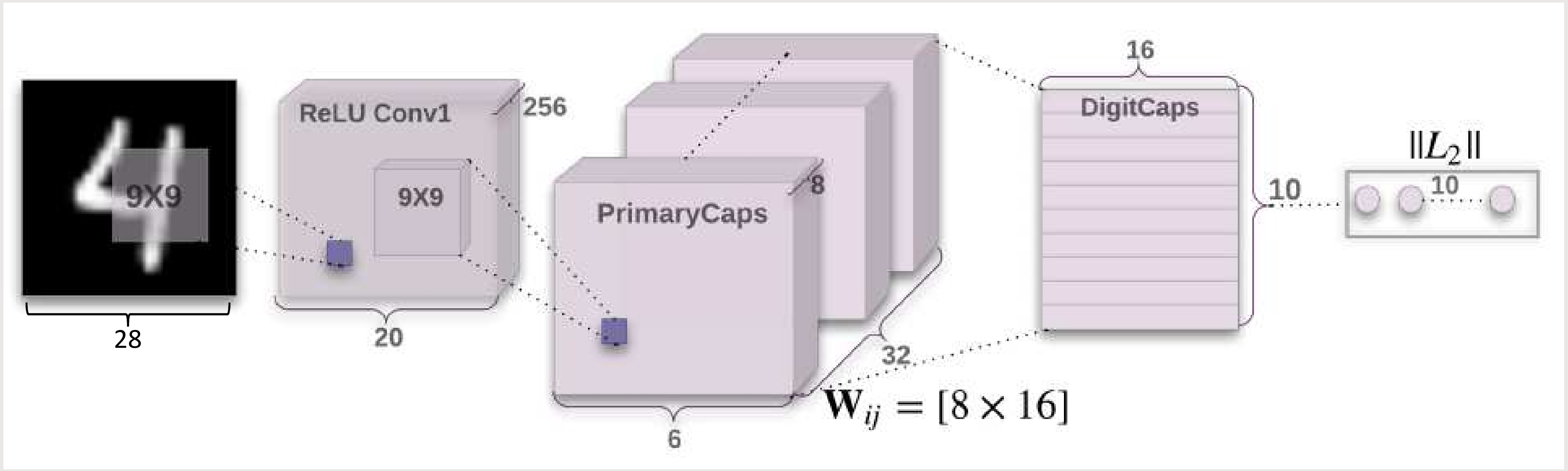
```
1: procedure ROUTING($\hat{u}_{j|i}$, $r$, $l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3:     for $r$ iterations do (Suggesting r as 3)
4:         for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ 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

Credit: Sabour et al., 2017



Credit: Pechyonkin, M. (2017)

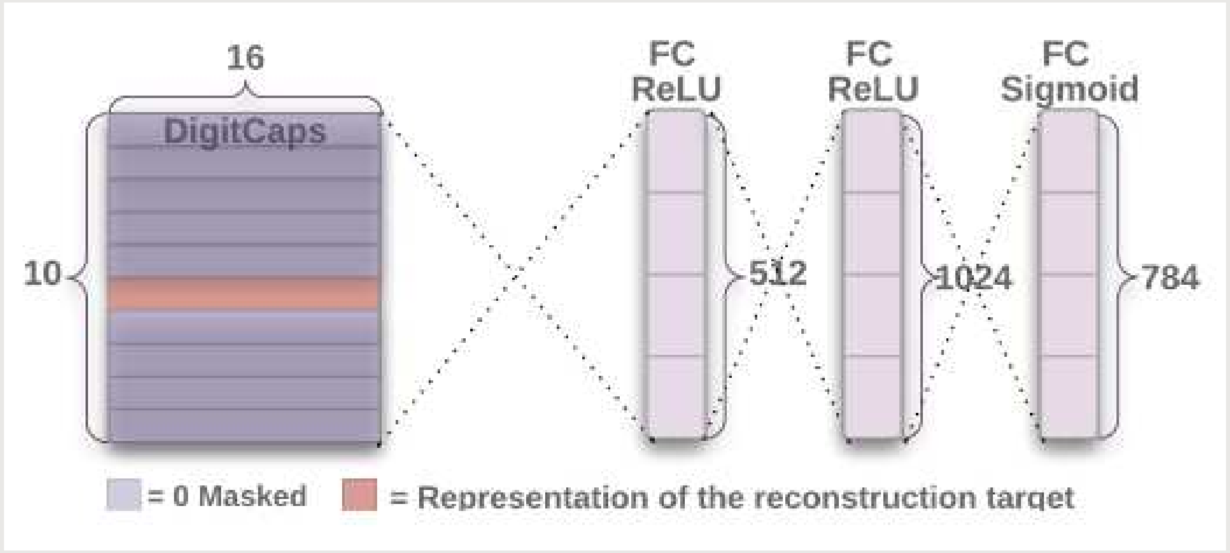
Architecture of capsule network



- Conv.: 256 kernels with sizes of 9x9x1 and stride 1, followed by ReLU activation, so the outputs with sizes of 20x20x256
- PrimaryCaps: 9x9x256 convolutional kernels (with stride 2), so the outputs with sizes of 6x6x8x32 (32 capsules), then reshape as u_i with sizes of (6x6x32)x8
- DigitCaps: v_j with sizes of 10x16, then the norm of each v_j is the possibility of prediction

Credit: Sabour et al., 2017

Reconstruction network (Optional)

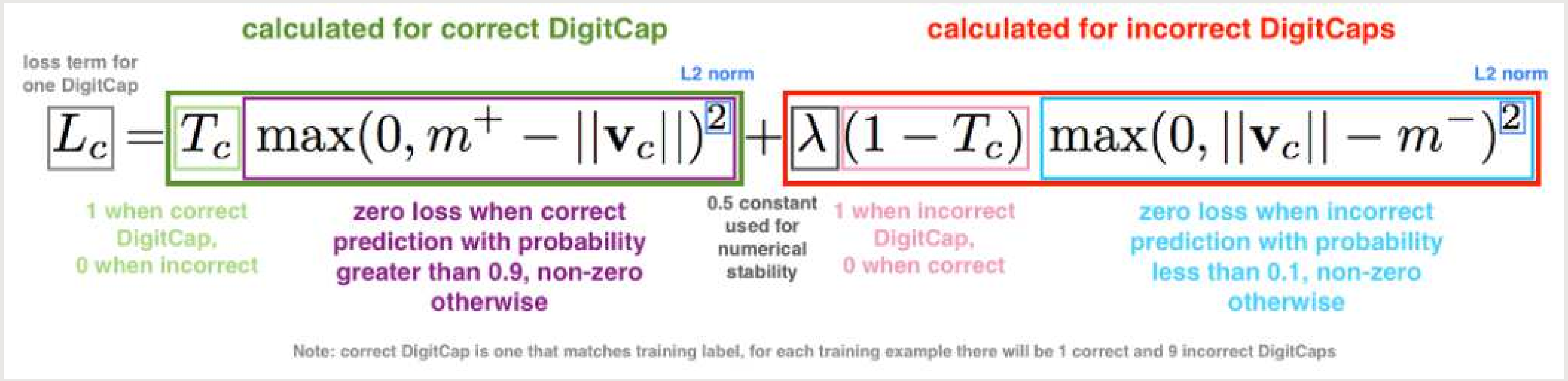


- Fully Connected: mask with the target from DigitCaps, then FC layer with sizes of 512, followed by ReLU activation
- Fully Connected: with sizes of 1024, followed by ReLU activation
- Fully Connected: with sizes of 784, followed by Sigmoid activation, then compared with the input (reshaped as 784x1)

Credit: Sabour et al., 2017

Loss function

- **Margin loss**



calculated for correct DigitCap
calculated for incorrect DigitCaps
loss term for one DigitCap
L2 norm
L2 norm
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

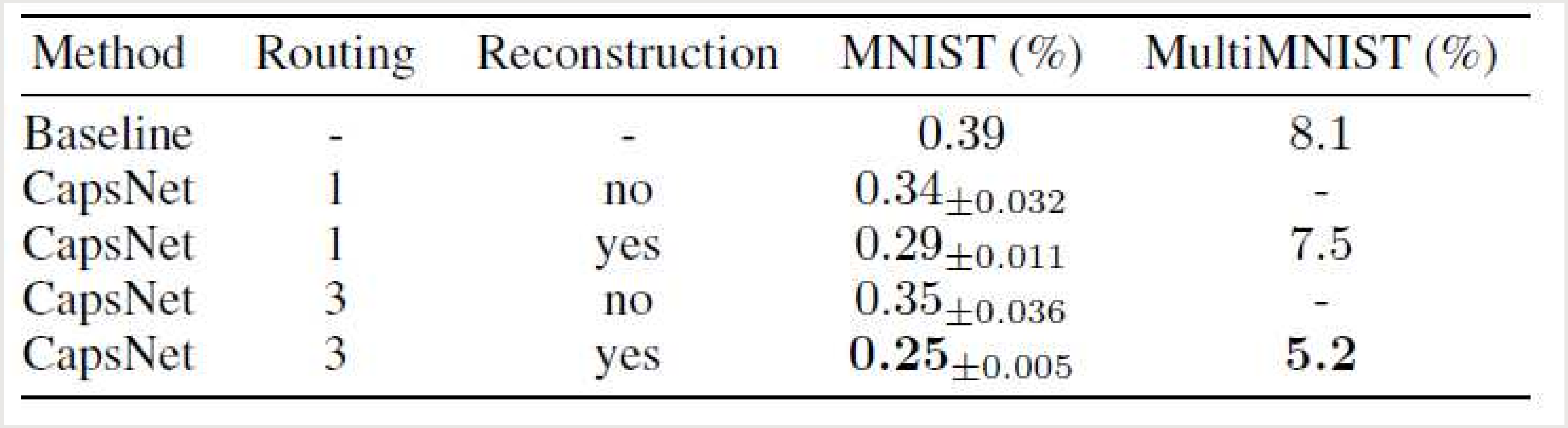
$$\boxed{L_c} = \boxed{T_c}\,\boxed{\max(0, m^+ - ||\mathbf{v}_c||)}^{\boxed{2}} + \boxed{\lambda}\,\boxed{(1 - T_c)}\,\boxed{\max(0, ||\mathbf{v}_c|| - m^-)}^{\boxed{2}}$$

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

- **Reconstruction loss (Optional): MSE**
- **Total loss = Margin loss + λ*reconstruction loss (suggesting λ as 0.0005)**

Credit: Sabour et al., 2017

Performance comparison



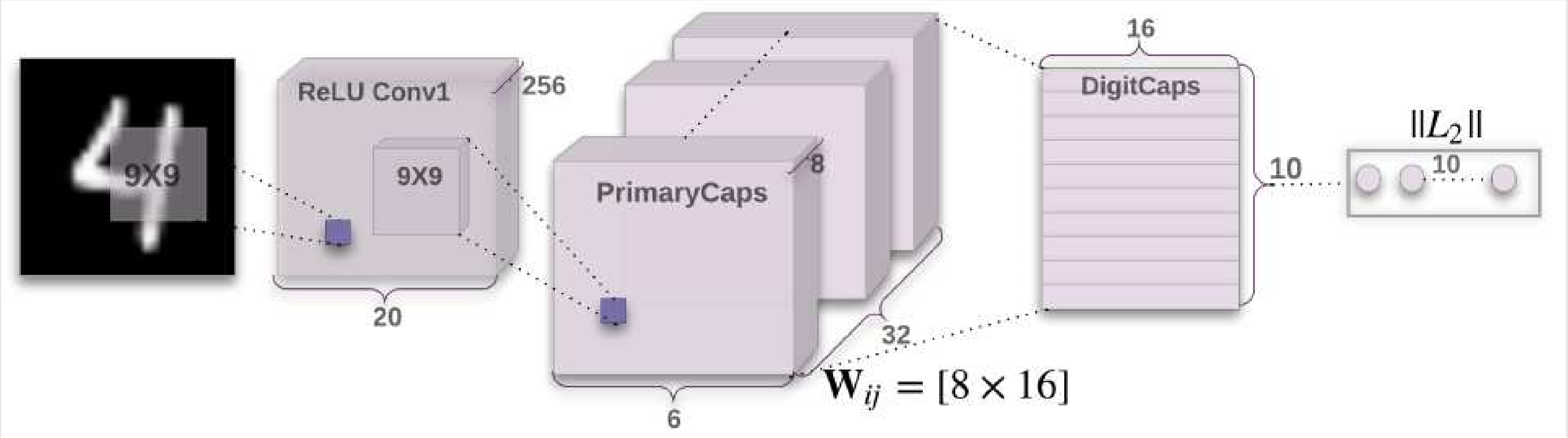
Method	Routing	Reconstruction	MNIST (%)	MultiMNIST (%)
Baseline	-	-	0.39	8.1
CapsNet	1	no	$0.34_{\pm 0.032}$	-
CapsNet	1	yes	$0.29_{\pm 0.011}$	7.5
CapsNet	3	no	$0.35_{\pm 0.036}$	-
CapsNet	3	yes	$\mathbf{0.25}_{\pm 0.005}$	**5.2**

Credit: Sabour et al., 2017

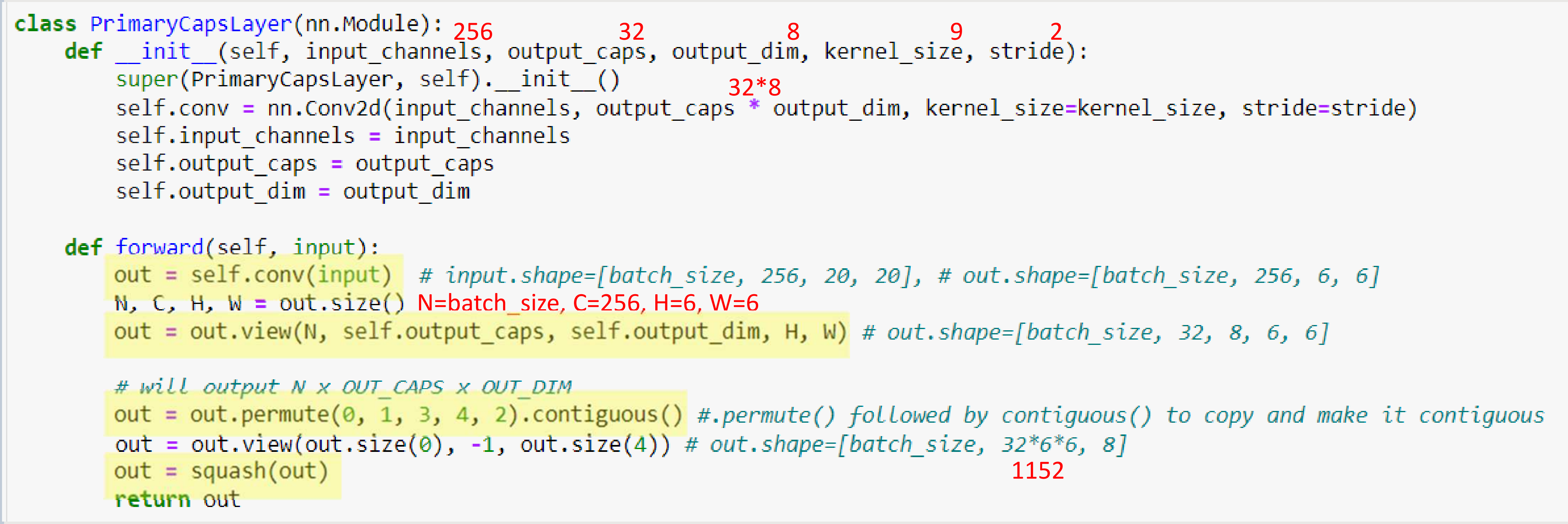
Example of codes

```
class CapsNet(nn.Module):
    def __init__(self, routing_iterations, n_classes=10):
        super(CapsNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 256, kernel_size=9, stride=1)
        self.primaryCaps = PrimaryCapsLayer(256, 32, 8, kernel_size=9, stride=2)  # outputs 6*6
        self.num_primaryCaps = 32 * 6 * 6
        routing_module = AgreementRouting(self.num_primaryCaps, n_classes, routing_iterations)
        self.digitCaps = CapsLayer(self.num_primaryCaps, 8, n_classes, 16, routing_module)
                                                                  10

    def forward(self, input):
        x = self.conv1(input) # input.shape=[batch_size,1,28,28]
        x = F.relu(x)
        x = self.primaryCaps(x)
        x = self.digitCaps(x) # x.shape=[batch_size, 10, 16]
        probs = x.pow(2).sum(dim=2).sqrt() # probs.shape=[batch_size,10]
        return x, probs
```



Example of codes

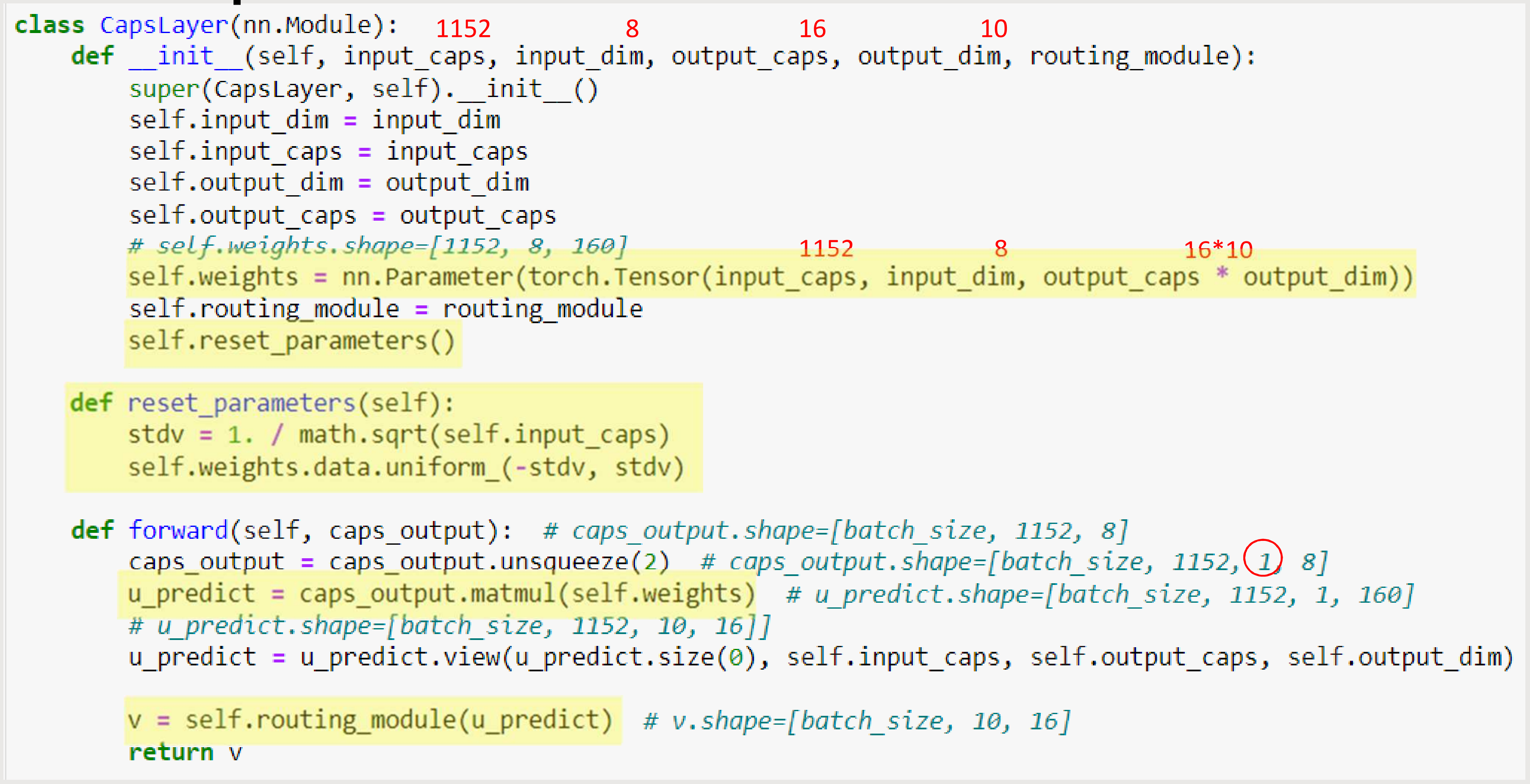


```
class PrimaryCapsLayer(nn.Module):  256            32            8          9          2
    def __init__(self, input_channels, output_caps, output_dim, kernel_size, stride):
        super(PrimaryCapsLayer, self).__init__()           32*8
        self.conv = nn.Conv2d(input_channels, output_caps * output_dim, kernel_size=kernel_size, stride=stride)
        self.input_channels = input_channels
        self.output_caps = output_caps
        self.output_dim = output_dim

    def forward(self, input):
        out = self.conv(input)  # input.shape=[batch_size, 256, 20, 20], # out.shape=[batch_size, 256, 6, 6]
        N, C, H, W = out.size() N=batch_size, C=256, H=6, W=6
        out = out.view(N, self.output_caps, self.output_dim, H, W) # out.shape=[batch_size, 32, 8, 6, 6]

        # will output N x OUT_CAPS x OUT_DIM
        out = out.permute(0, 1, 3, 4, 2).contiguous() #.permute() followed by contiguous() to copy and make it contiguous
        out = out.view(out.size(0), -1, out.size(4)) # out.shape=[batch_size, 32*6*6, 8]
        out = squash(out)                                                      1152
        return out
```

Example of codes



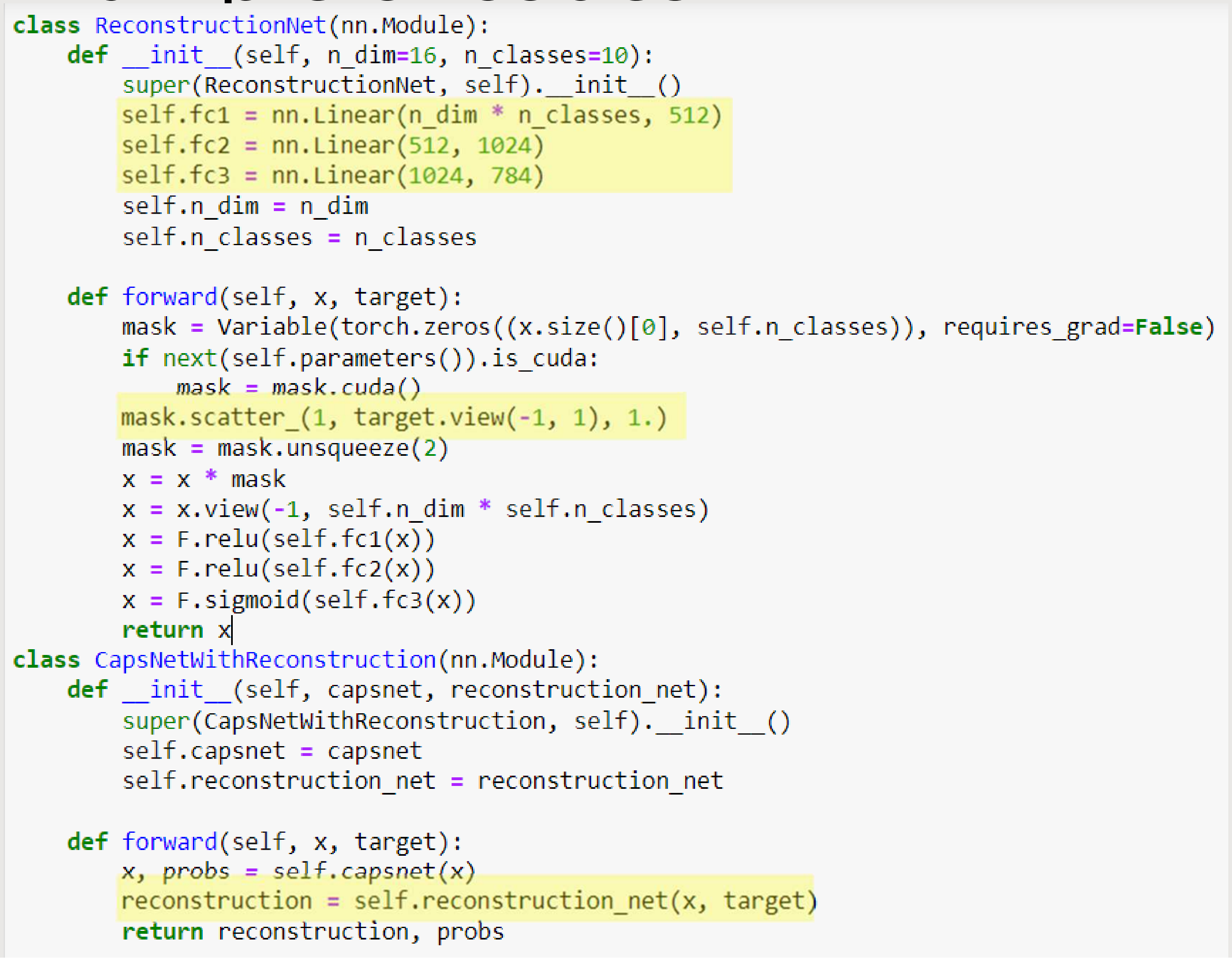
```
class CapsLayer(nn.Module):   1152           8             16            10
    def __init__(self, input_caps, input_dim, output_caps, output_dim, routing_module):
        super(CapsLayer, self).__init__()
        self.input_dim = input_dim
        self.input_caps = input_caps
        self.output_dim = output_dim
        self.output_caps = output_caps
        # self.weights.shape=[1152, 8, 160]            1152         8           16*10
        self.weights = nn.Parameter(torch.Tensor(input_caps, input_dim, output_caps * output_dim))
        self.routing_module = routing_module
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.input_caps)
        self.weights.data.uniform_(-stdv, stdv)

    def forward(self, caps_output):  # caps_output.shape=[batch_size, 1152, 8]
        caps_output = caps_output.unsqueeze(2)  # caps_output.shape=[batch_size, 1152, 1, 8]
        u_predict = caps_output.matmul(self.weights)  # u_predict.shape=[batch_size, 1152, 1, 160]
        # u_predict.shape=[batch_size, 1152, 10, 16]]
        u_predict = u_predict.view(u_predict.size(0), self.input_caps, self.output_caps, self.output_dim)

        v = self.routing_module(u_predict)  # v.shape=[batch_size, 10, 16]
        return v
```

Example of codes

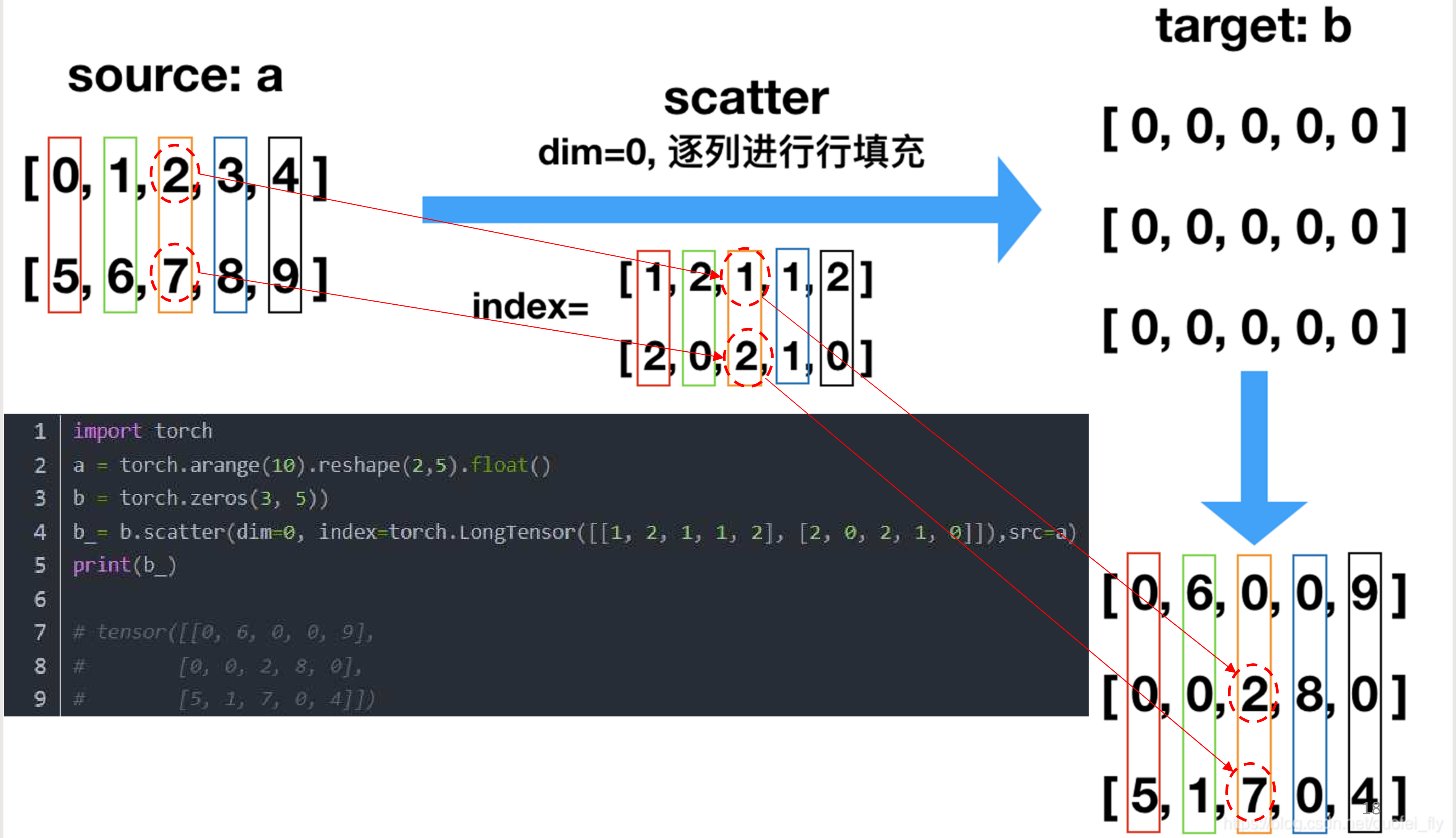


```python
class ReconstructionNet(nn.Module):
    def __init__(self, n_dim=16, n_classes=10):
        super(ReconstructionNet, self).__init__()
        self.fc1 = nn.Linear(n_dim * n_classes, 512)
        self.fc2 = nn.Linear(512, 1024)
        self.fc3 = nn.Linear(1024, 784)
        self.n_dim = n_dim
        self.n_classes = n_classes

    def forward(self, x, target):
        mask = Variable(torch.zeros((x.size()[0], self.n_classes)), requires_grad=False)
        if next(self.parameters()).is_cuda:
            mask = mask.cuda()
        mask.scatter_(1, target.view(-1, 1), 1.)
        mask = mask.unsqueeze(2)
        x = x * mask
        x = x.view(-1, self.n_dim * self.n_classes)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.sigmoid(self.fc3(x))
        return x
class CapsNetWithReconstruction(nn.Module):
    def __init__(self, capsnet, reconstruction_net):
        super(CapsNetWithReconstruction, self).__init__()
        self.capsnet = capsnet
        self.reconstruction_net = reconstruction_net

    def forward(self, x, target):
        x, probs = self.capsnet(x)
        reconstruction = self.reconstruction_net(x, target)
        return reconstruction, probs
```

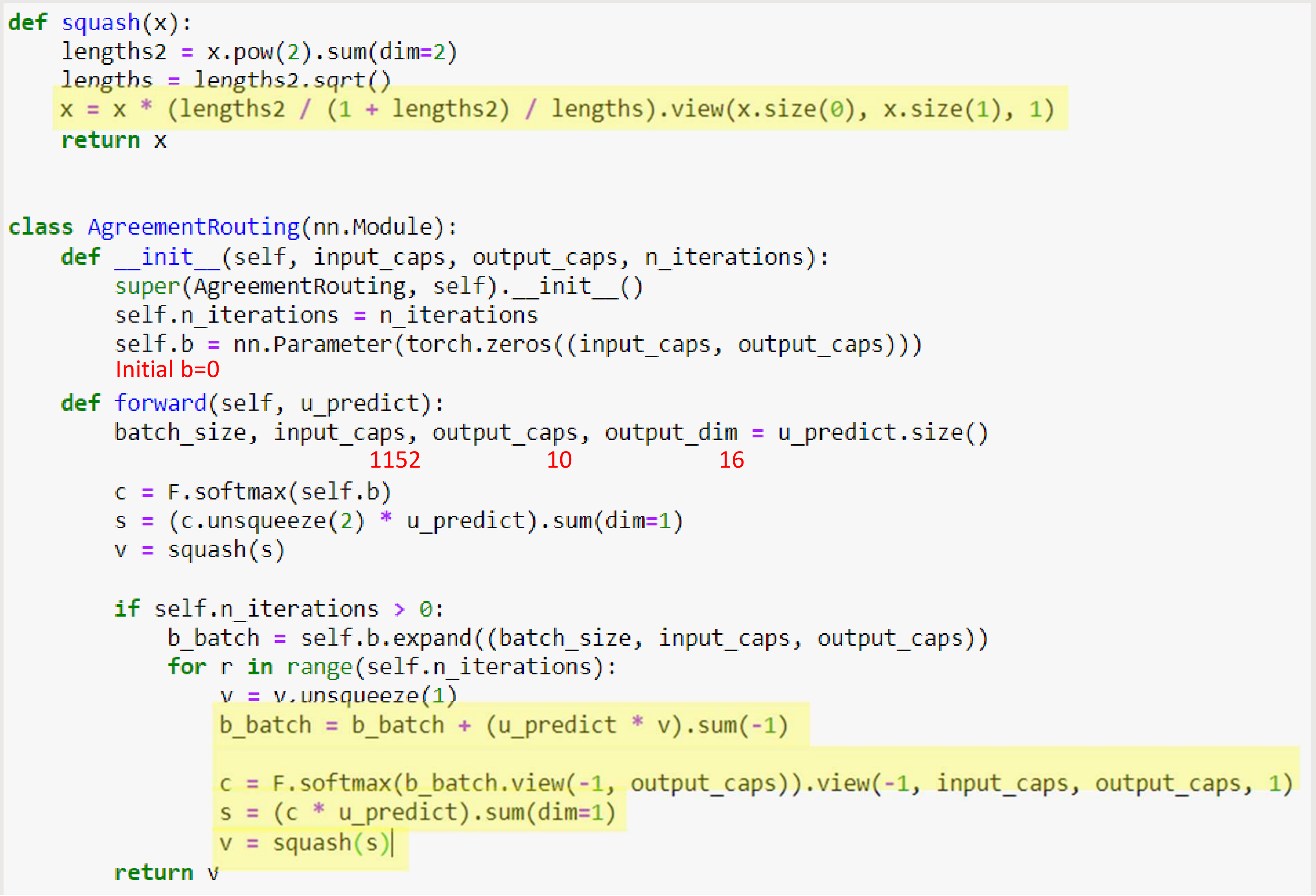
Pytorch scatter function



```python
import torch
a = torch.arange(10).reshape(2,5).float()
b = torch.zeros(3, 5))
b_= b.scatter(dim=0, index=torch.LongTensor([[1, 2, 1, 1, 2], [2, 0, 2, 1, 0]]),src=a)
print(b_)

# tensor([[0, 6, 0, 0, 9],
#         [0, 0, 2, 8, 0],
#         [5, 1, 7, 0, 4]])
```

Example of codes



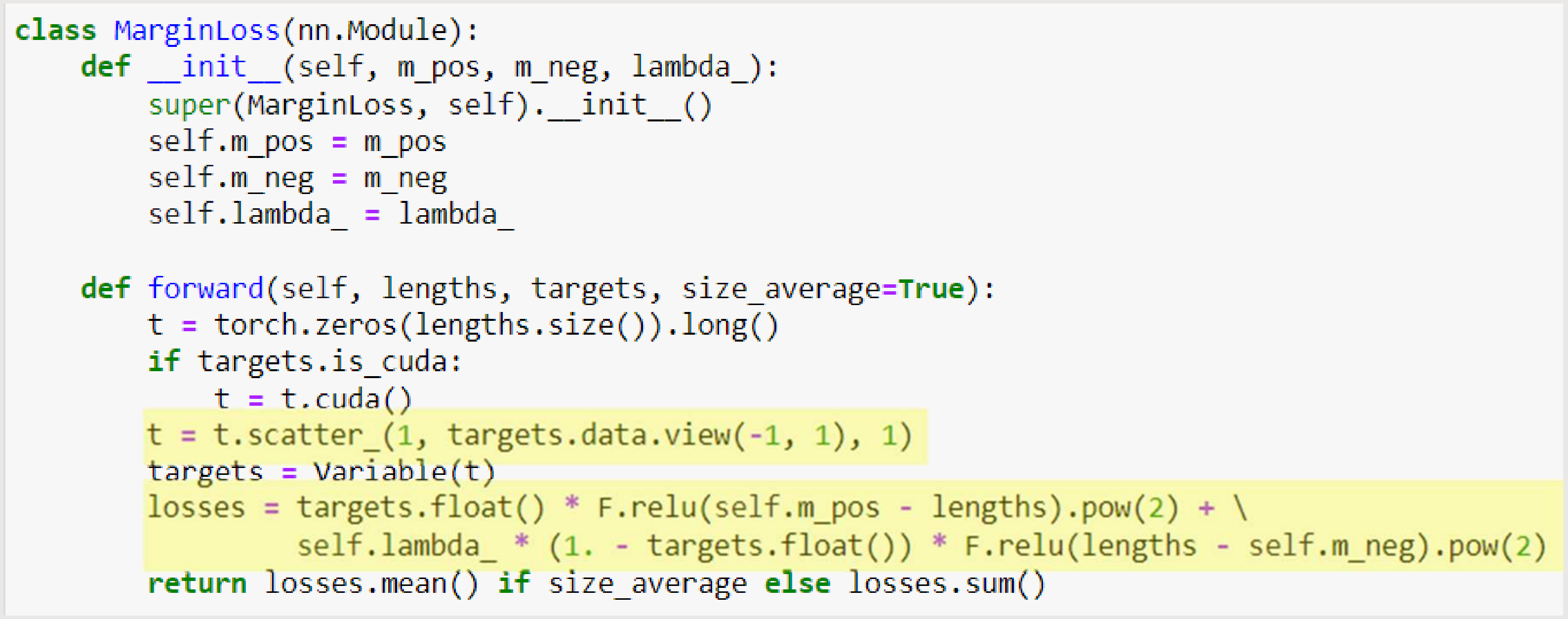
```
def squash(x):
    lengths2 = x.pow(2).sum(dim=2)
    lengths = lengths2.sqrt()
    x = x * (lengths2 / (1 + lengths2) / lengths).view(x.size(0), x.size(1), 1)
    return x


class AgreementRouting(nn.Module):
    def __init__(self, input_caps, output_caps, n_iterations):
        super(AgreementRouting, self).__init__()
        self.n_iterations = n_iterations
        self.b = nn.Parameter(torch.zeros((input_caps, output_caps)))
        Initial b=0
    def forward(self, u_predict):
        batch_size, input_caps, output_caps, output_dim = u_predict.size()
                           1152          10             16
        c = F.softmax(self.b)
        s = (c.unsqueeze(2) * u_predict).sum(dim=1)
        v = squash(s)

        if self.n_iterations > 0:
            b_batch = self.b.expand((batch_size, input_caps, output_caps))
            for r in range(self.n_iterations):
                v = v.unsqueeze(1)
                b_batch = b_batch + (u_predict * v).sum(-1)

                c = F.softmax(b_batch.view(-1, output_caps)).view(-1, input_caps, output_caps, 1)
                s = (c * u_predict).sum(dim=1)
                v = squash(s)
        return v
```

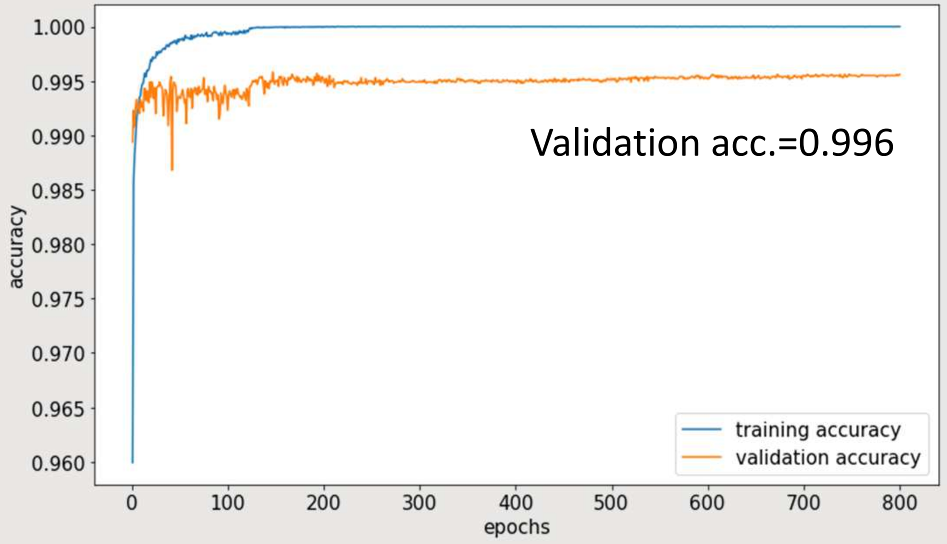
Example of codes



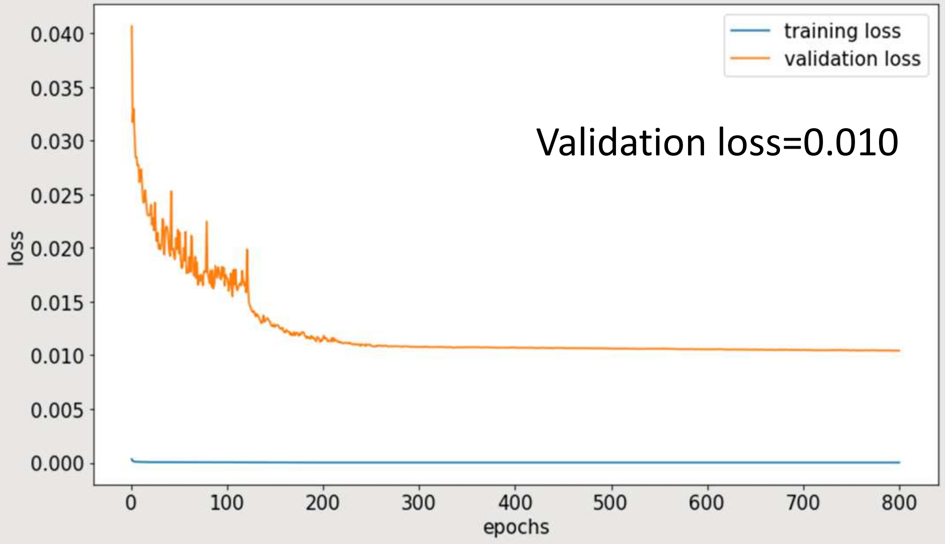
```python
class MarginLoss(nn.Module):
    def __init__(self, m_pos, m_neg, lambda_):
        super(MarginLoss, self).__init__()
        self.m_pos = m_pos
        self.m_neg = m_neg
        self.lambda_ = lambda_

    def forward(self, lengths, targets, size_average=True):
        t = torch.zeros(lengths.size()).long()
        if targets.is_cuda:
            t = t.cuda()
        t = t.scatter_(1, targets.data.view(-1, 1), 1)
        targets = Variable(t)
        losses = targets.float() * F.relu(self.m_pos - lengths).pow(2) + \
                 self.lambda_ * (1. - targets.float()) * F.relu(lengths - self.m_neg).pow(2)
        return losses.mean() if size_average else losses.sum()
```

Result of codes



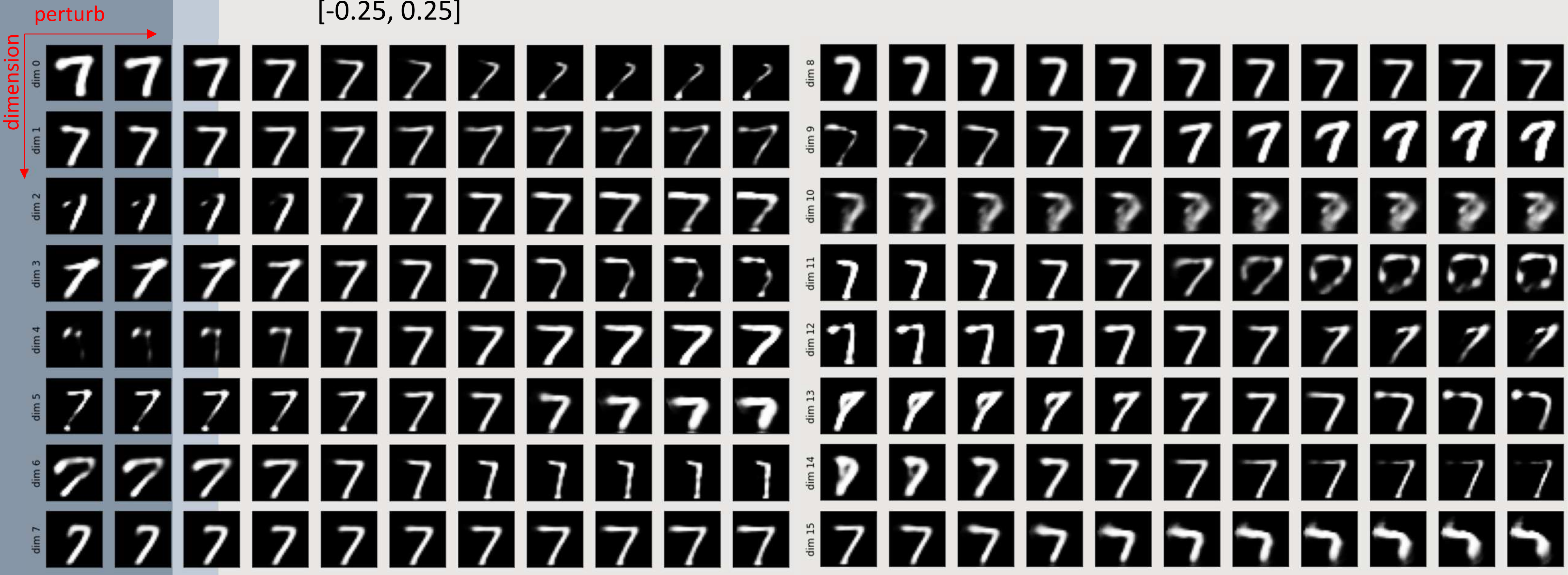
Validation acc.=0.996



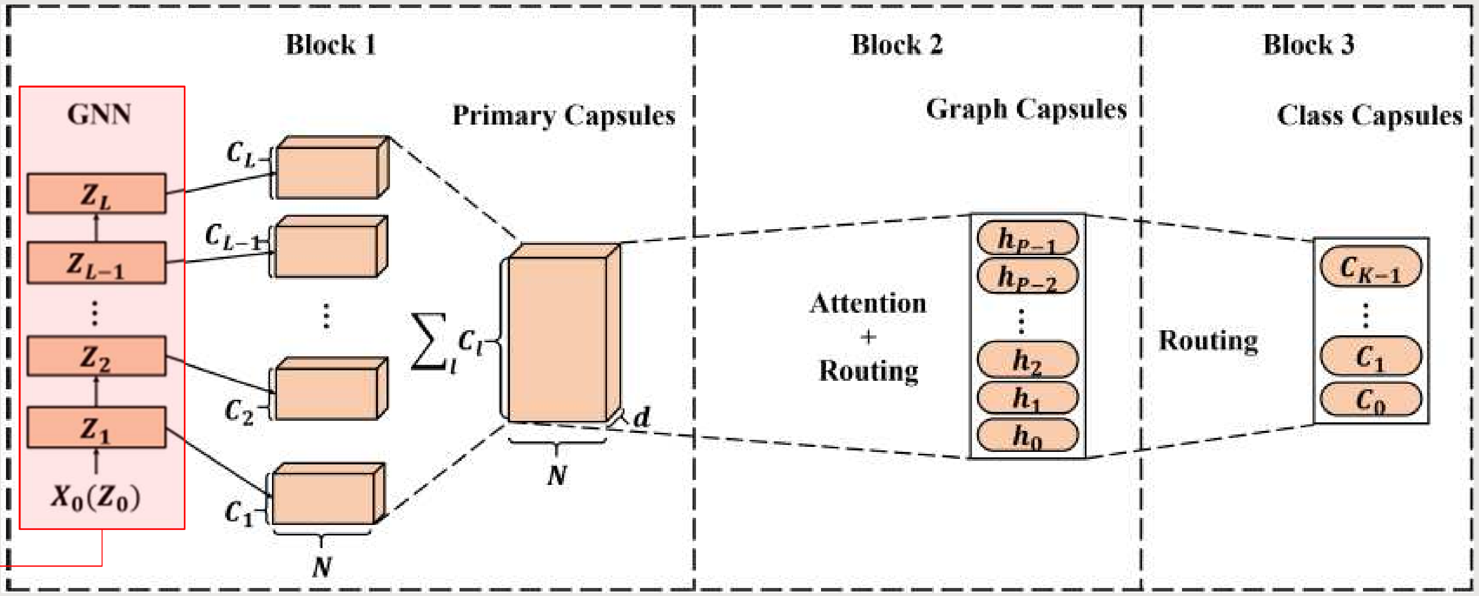
Validation loss=0.010

Result of codes

- Perturb the DigitCaps to manipulate the reconstruction by intervals of 0.05 in the range of [-0.25, 0.25]



Capsule Graph Neural Network

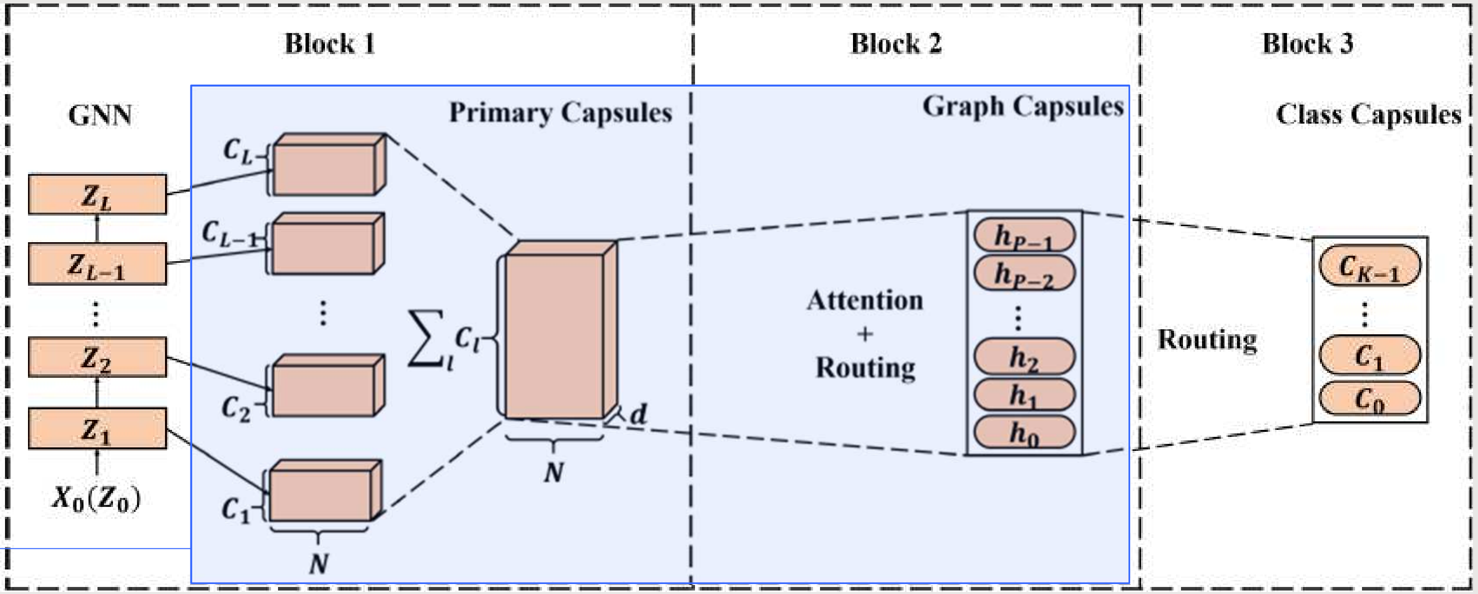


$$z_j^{l+1} = f\left(\sum_i \widetilde{D}^{\frac{-1}{2}} \tilde{A} \widetilde{D}^{\frac{-1}{2}} Z_i^l W_{ij}^l\right) \text{(GCN)}$$

z_j^{l+1}: node feature at the layer $l+1$, $f(\cdot)$: activation function, $\widetilde{D}$: degree matrix, $\tilde{A}$: adjacency matrix, W_{ij}^l: trainable weight matrix

Credit:Xinyi & Chen, 2018

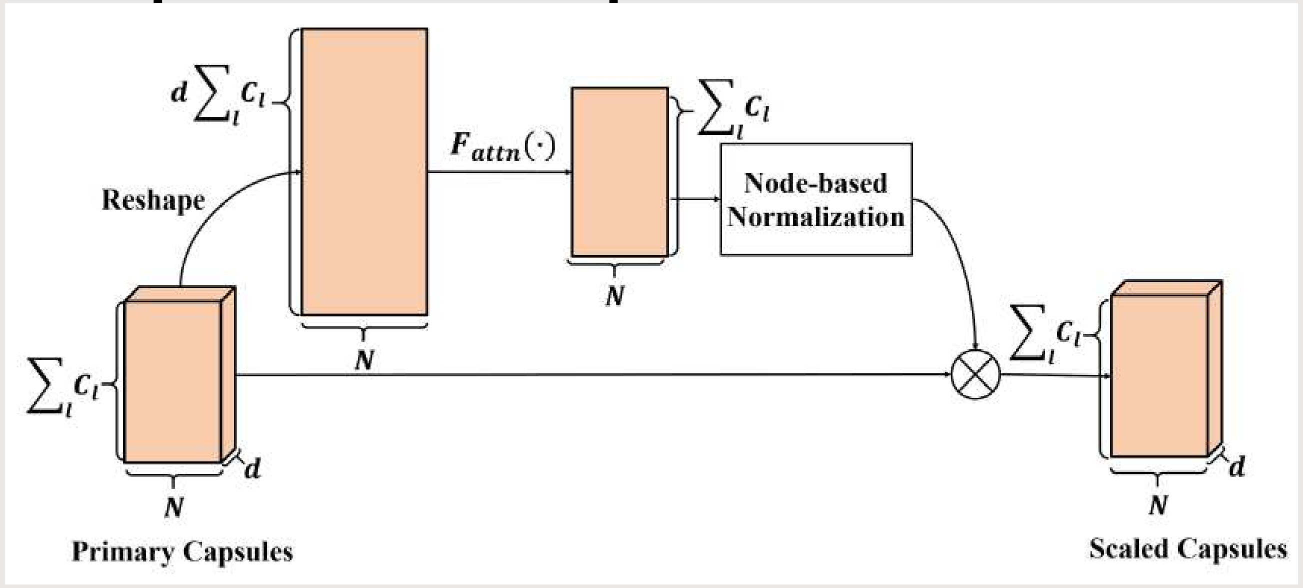
Capsule Graph Neural Network



N: set of node capsules,
C_l: number of channels at the layer l,
d: dimension
h_p: graph capsules

Credit:Xinyi & Chen, 2018

Capsule Graph Neural Network



$$scaled\left(s_{(n,i)}\right) = \frac{F_{attn}(\tilde{s}_n)_i}{\sum_n F_{attn}(\tilde{s}_n)_i} s_{(n,i)}$$

$\tilde{s}_n$: obtained by concatenating all capsules of the node n,

$s_{(n,i)}$: represents the i th capsule of the node n,

$F_{attn}(\tilde{s}_n)$: the generated attention value

Credit:Xinyi & Chen, 2018

Reference

- Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. *Advances in neural information processing systems, 30.*
- Pechyonkin, M. (2017) Understanding Hinton’ s Capsule Networks, retrieved May 19, 2022, from: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-iii-dynamic-routing-between-capsules-349f6d30418
- Lee, H. Y. (2017). Capsule, Deep Learning Online Course, retrieved May 19, 2022, from: https://www.bilibili.com/video/BV1Qx411V75M/
- Géron, A. (2017). Capsule Networks (CapsNets) – Tutorial, retrieved May 19, 2022, from: https://www.youtube.com/watch?v=pPN8d0E3900
- Bielski, A. (2019). Dynamic Routing Between Capsules - PyTorch implementation, retrieved May 19, 2022, from: https://github.com/adambielski/CapsNet-pytorch
- Xinyi, Z., & Chen, L. (2018). Capsule graph neural network. International conference on learning representations,
- Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907.*

AI 研究中心