

Dynamic Routing Between Capsules

Sara Sabour, Nicholas Frosst, Geoffrey E Hinton

Thinking in Vectors

Reducing Neural Network Complexity
With Vector Math

Overview

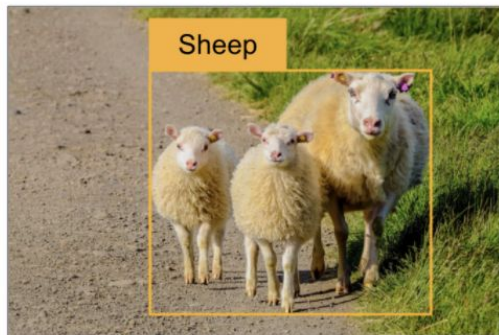
1. Problem & Background
2. How Capsules Work
3. Dynamic Routing
4. CapsNet Architecture
5. Experiments & Results
6. Discussion

— — —

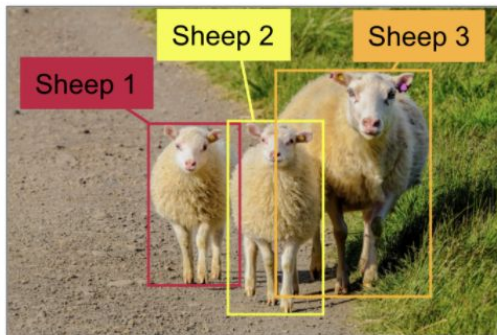
Problem & Background

The Problem: Human-Level Visual Perception

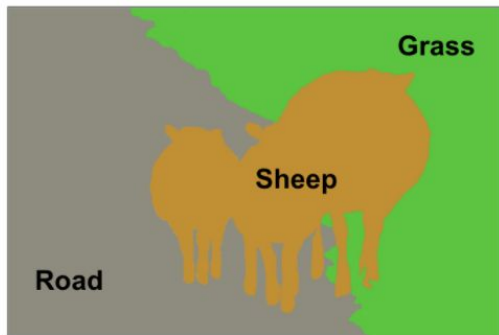
— — —



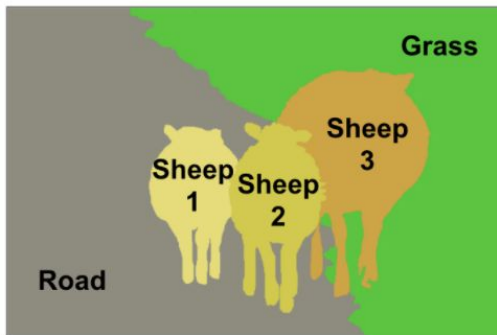
Classification + Localization



Object Detection



Semantic Segmentation



Instance Segmentation

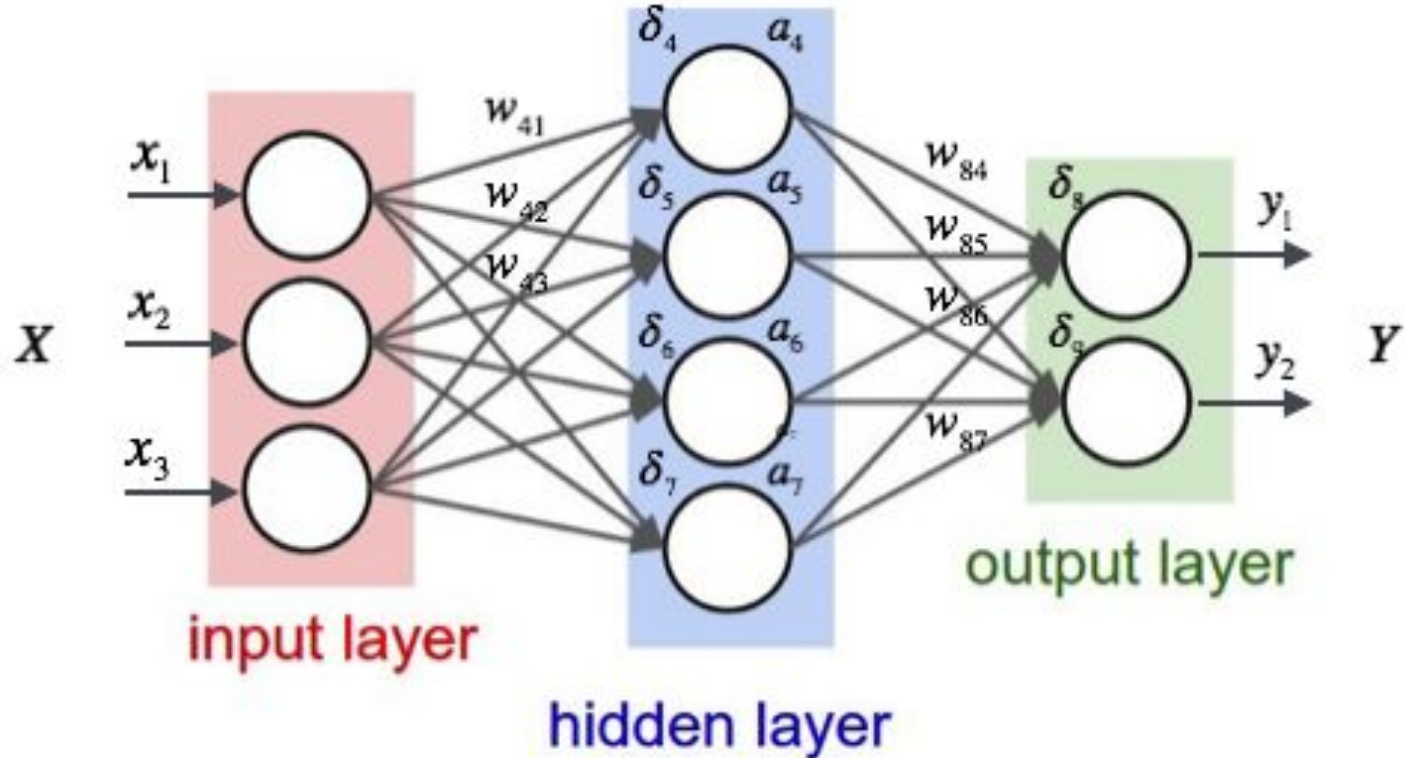


What is that? - Neural Networks

Simulated neurons arranged as a network.

- Mimic organic brains at a very small scale.
- Mathematically, a series of linear algebra calculations.
- Very efficient at runtime.
- Training usually takes a lot of time or special hardware.

What is that? - Neural Networks



What is that? - Vectors

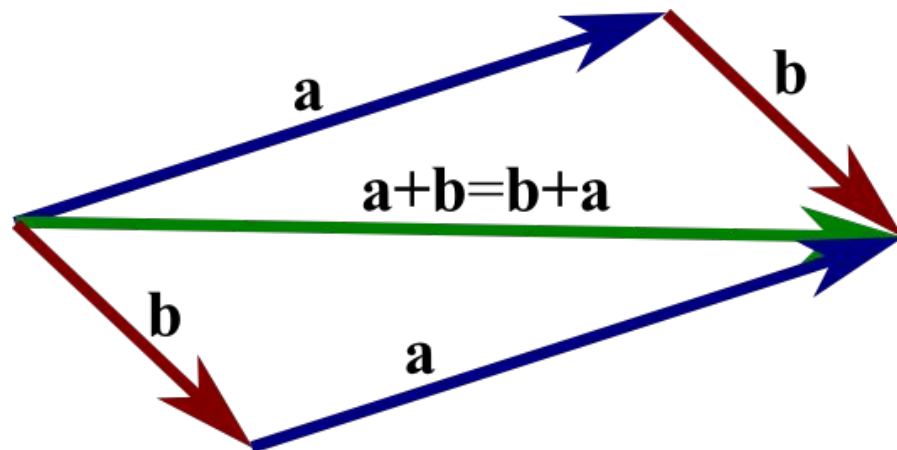
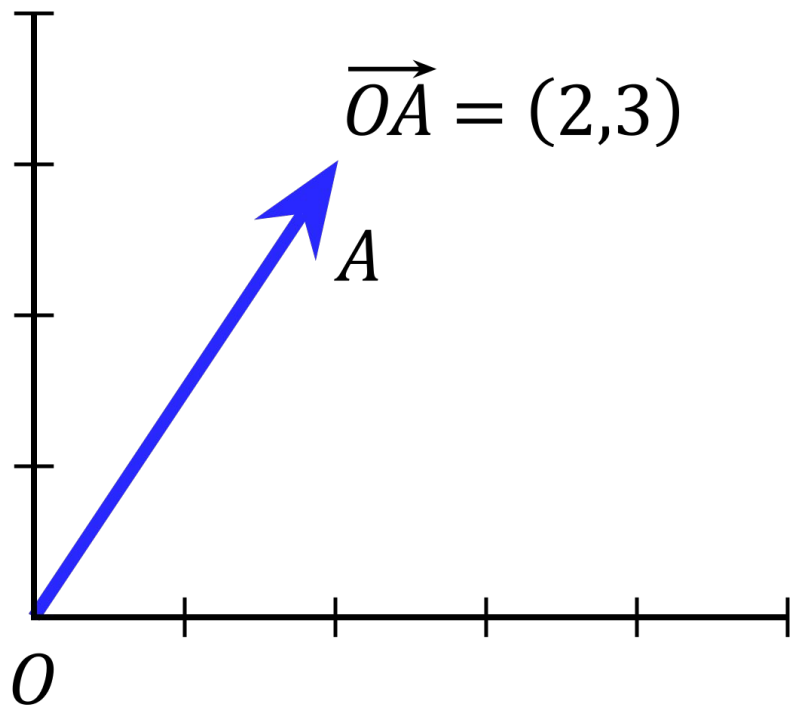
Mathematical quantity with both direction and magnitude.

- Quantity with a scalar value assigned to each dimension.
- The building block of linear algebra.

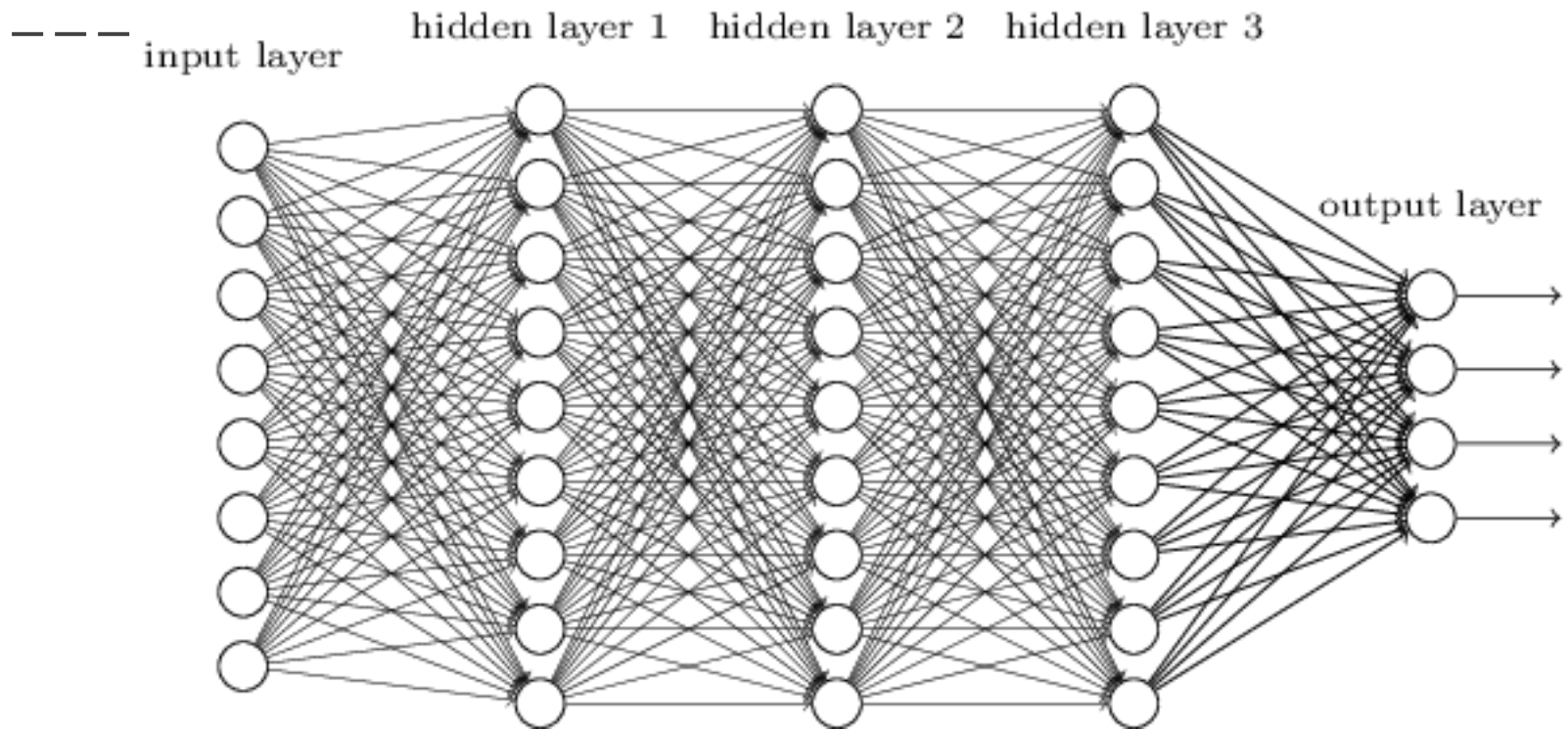
For example,

- $(10, 21, 43) \leftarrow$ a 3D vector.
- $(10, 0, 0, 0) \leftarrow$ a 4D vector.

Vectors Illustrated



Traditional (Non-Capsule) Neural Networks

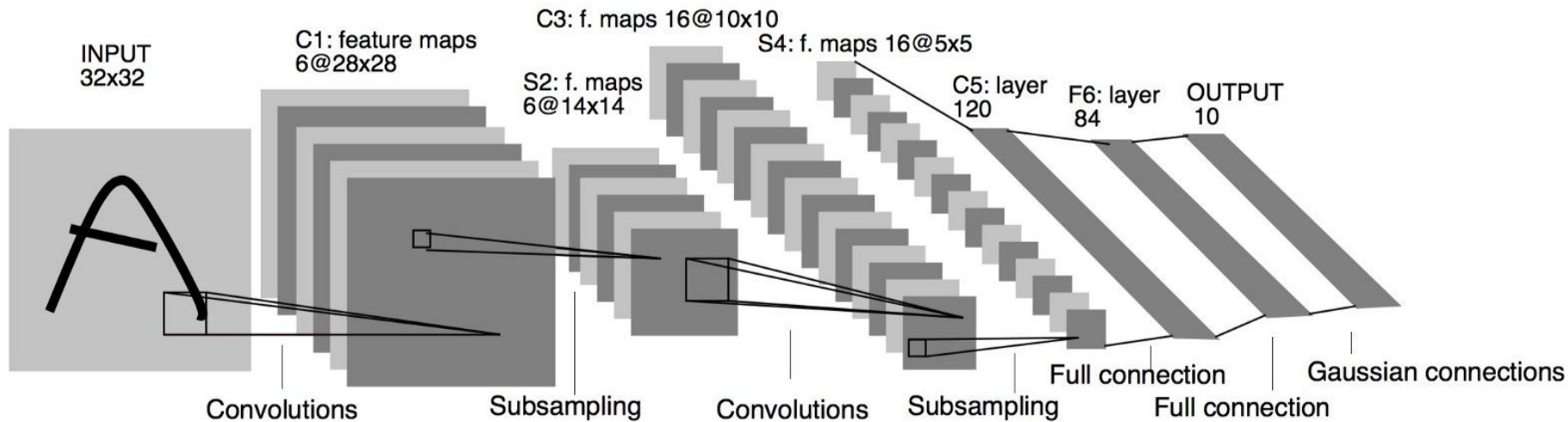


Traditional (Non-Capsule) Neural Networks

— — —

- Neurons “activate” independently.
- Requires very deep networks (10+ layers) to solve really interesting problems.
- For perception problems, usually requires very large amounts of similar training data (augmented/“jittered”).

Convolutional Neural Networks

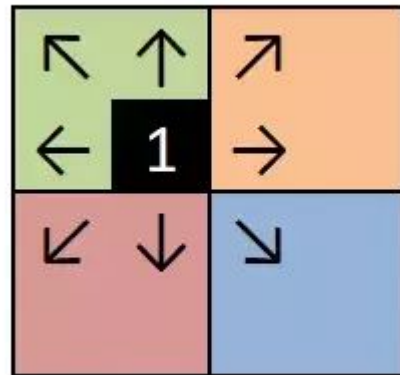


Convolutional Neural Networks

- Convolutional Layers break input data into smaller pieces which selectively cause neural activation.
- Pooling layers reduce these activations into invariant activations about the features present in the input data.
- Due to invariance between layers, CNNs require large amounts of training data to perform reasonably well.
- Useful networks tend to be very deep (10+ layers).

What is that? - Invariance

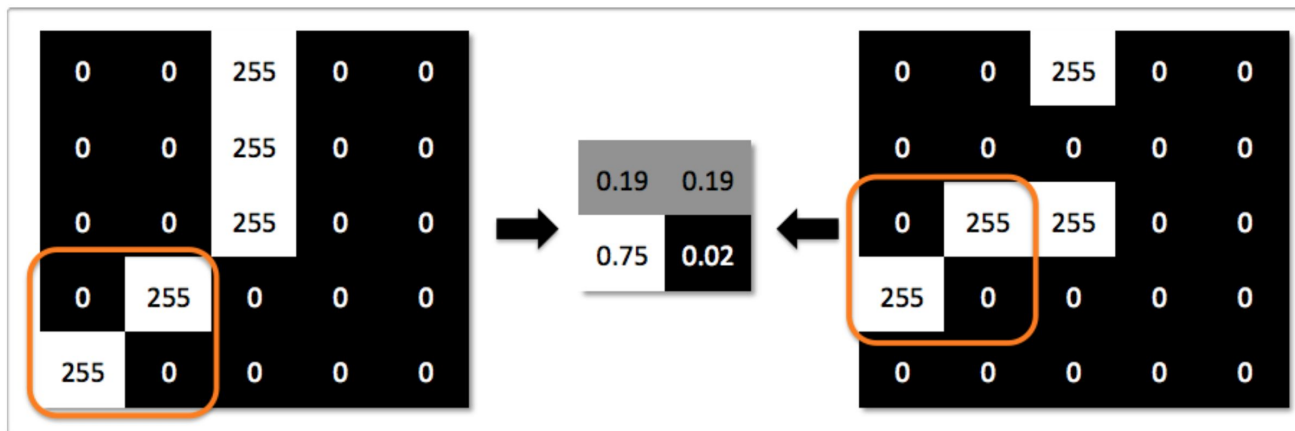
- Pooling layers trade resolution for performance.
- The output of a pooling layer is said to be “invariant” to transformations on the input.



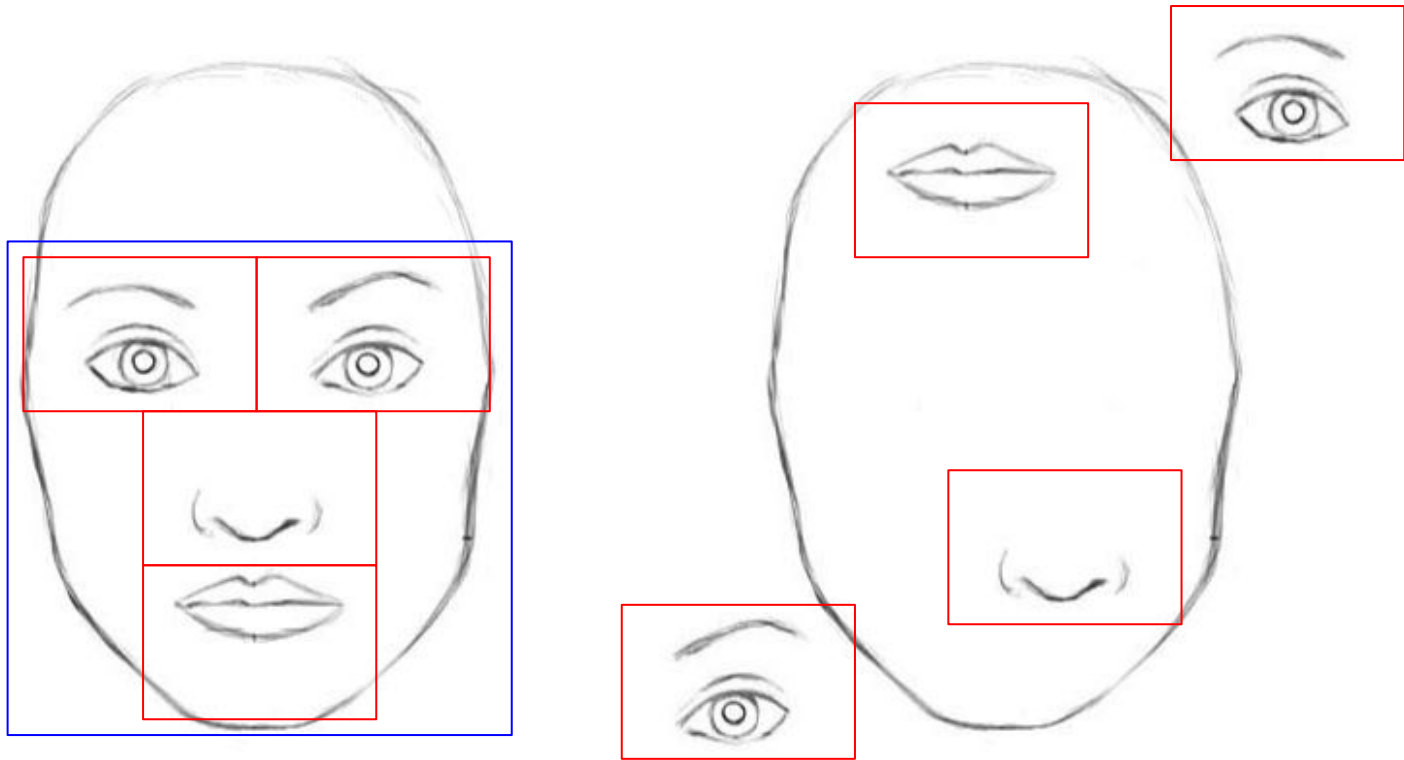
Invariance Illustrated

- Illustration of **local translation invariance**

- both images result in the same feature map after pooling/
subsampling



Invariance Illustrated



Ensuring Early Invariance is a Silly Goal

- Requirement: the final labels produced by the network must be viewpoint-invariant.
- Need not be satisfied by the ensuring invariance in neural activations within the network.
- Better goal: ensuring equivariance, which can be encoded by weights as well as activations in the network.

What is that? - Equivariance

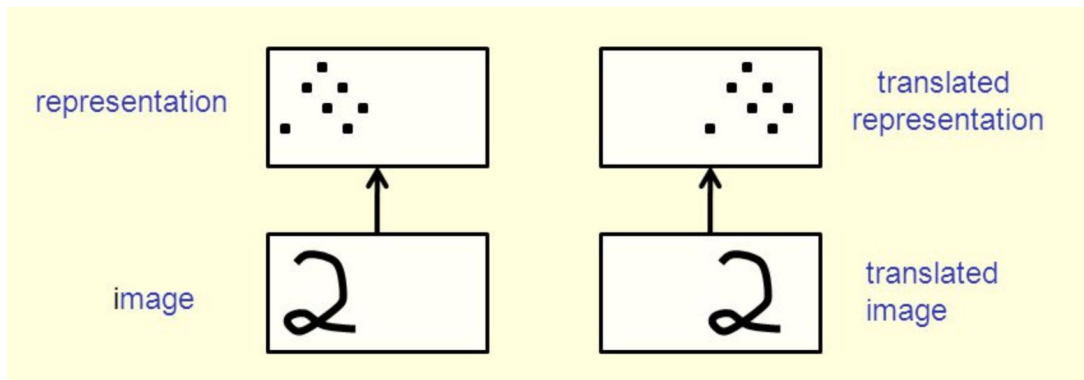
Changes in input lead to proportional changes in output

2 kinds of equivariance in perception networks:

- Place-coded - Changes in activation/routing.
- Rate-coded - Changes in output values.

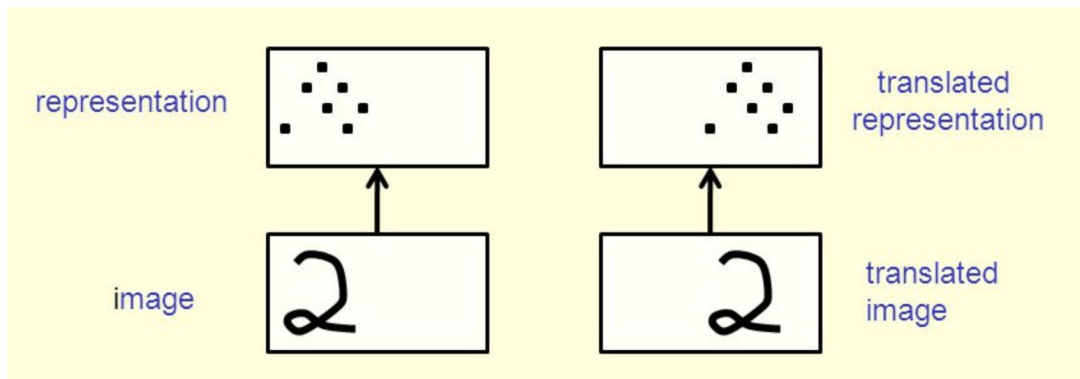
Place-Coded Equivariance

If a detected feature is moves dramatically, the network will activate a different route.

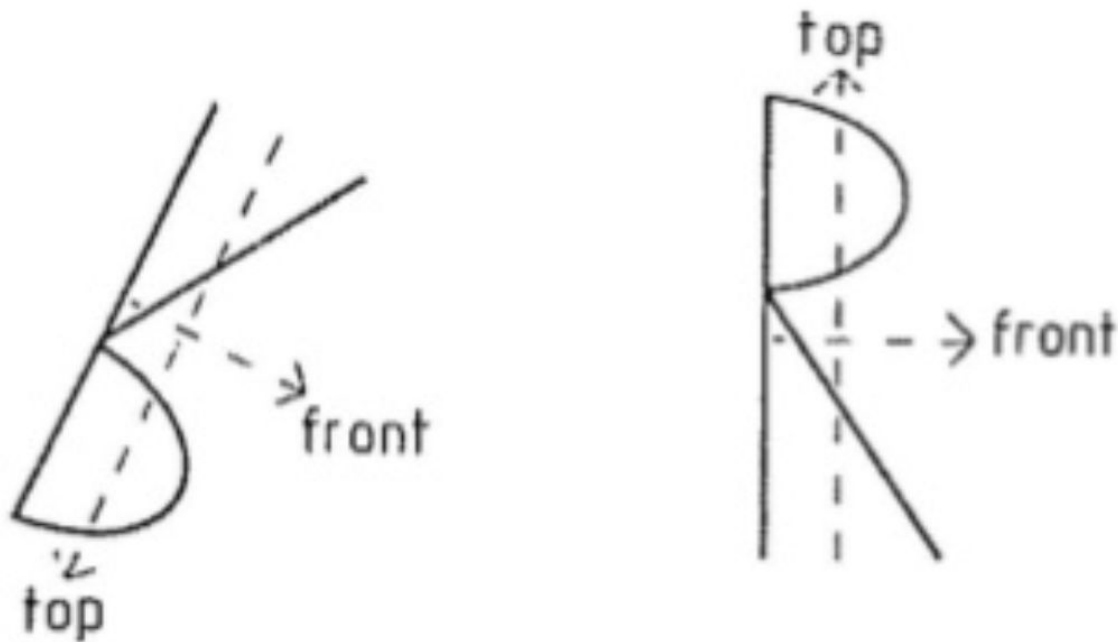


Rate-Coded Equivariance

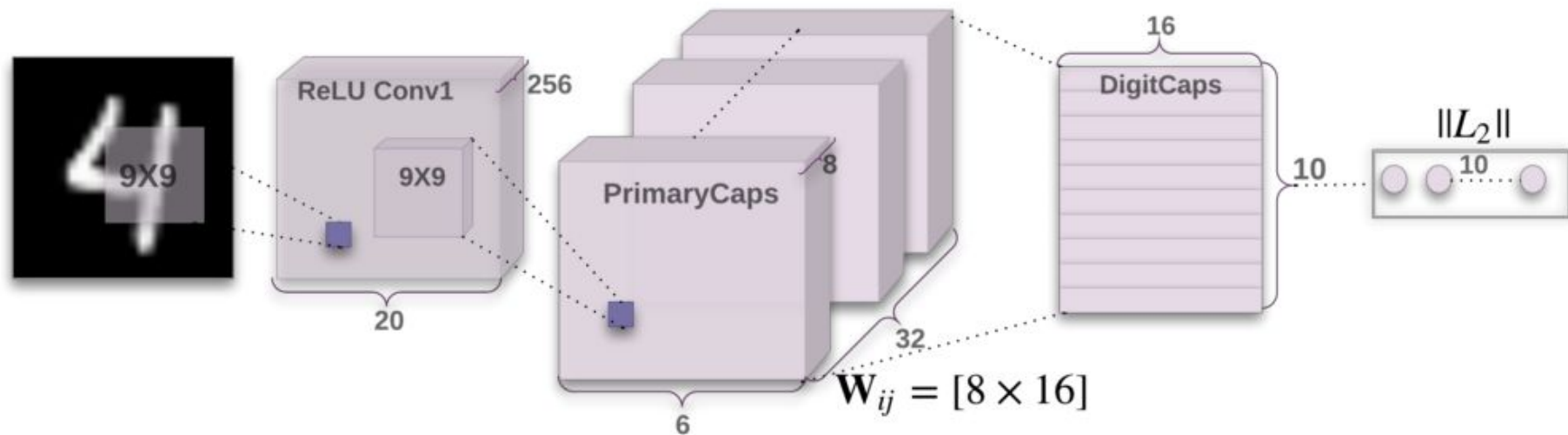
If a feature moves less dramatically, the same route will activate with different values.



Learning Intrinsic Coordinate Frames Demands Equivariance



This Paper's Solution - Convolutional Capsule Networks



How Capsules Work

Capsules are Introduced: “Transforming Auto-encoders”

— — —

“Instead of aiming for viewpoint invariance in the activities of “neurons” that use a single scalar output to summarize the activities of a local pool of replicated feature detectors, 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.”

A Simpler Take

— — —

- Capsules should do the bulk of computation internally.
- Capsules should pass their results as vectors.
- Probability of feature existence should be invariant.
- “Instantiation Parameters” should be equivariant.

Capsule Outputs as Vectors

- The magnitude of a capsule output represents the probability of existence of a feature.
- The orientation of the capsule output represents pose information (such as position, rotation, scale, color, and other deformations within its intrinsic coordinate frame).

Vector vs Scalar Output

Single neuron

7 0.9

1 0.3

Single capsule with
2 neurons

7 ↑

1 ↑

(Number rotated by 20°)

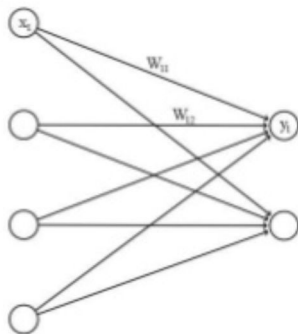
7 ↗

1 ↗

Capsules vs Neurons

— — ▪ Neuron

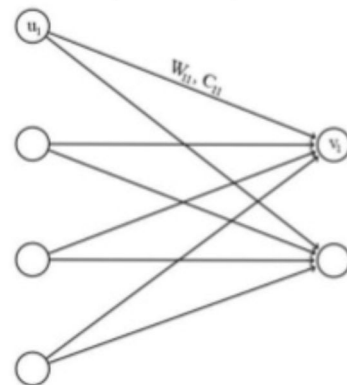
- The inputs of lower layers and the outputs of upper layers are **scalars**.
- After aggregating inputs multiplied by weights, nonlinearity function such as ReLU is applied.



$$z_j = \sum_i W_{ij} x_i$$
$$y_j = \text{ReLU}(z_j)$$

▪ Capsule

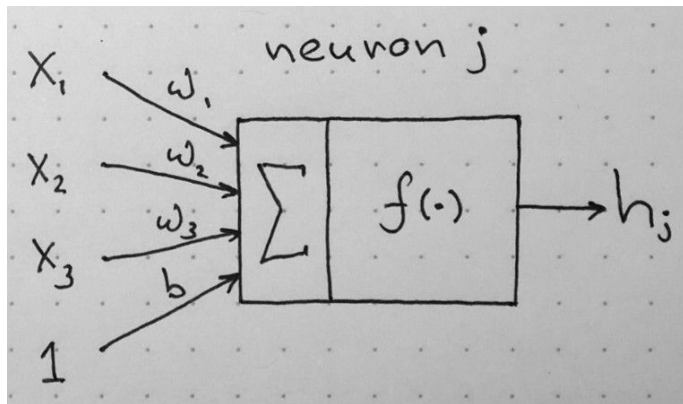
- The inputs of the lower layers and the outputs of upper layers are **vectors**.
- Instead of using nonlinearity function such as ReLU, we used **squashing** function.



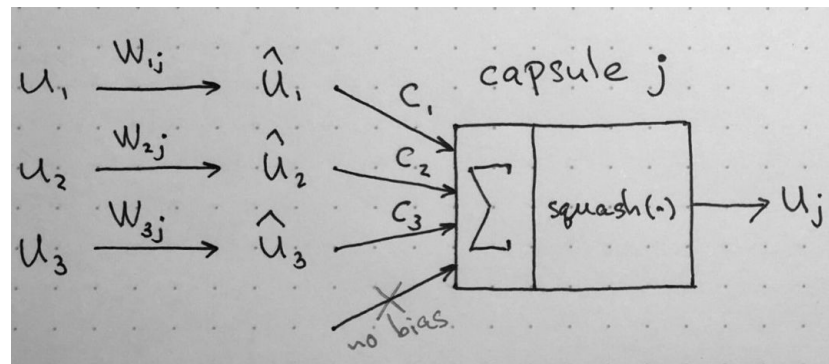
$$\hat{u}_{j|i} = W_{ij} u_i$$
$$s_j = \sum_i c_{ij} \hat{u}_{j|i}$$
$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|}$$

Neuron vs Capsule - Calculations

1. Scalar weighting of input scalars
2. Sum of weighted input scalars
3. Scalar->Scalar nonlinearity (e.g., ReLU)

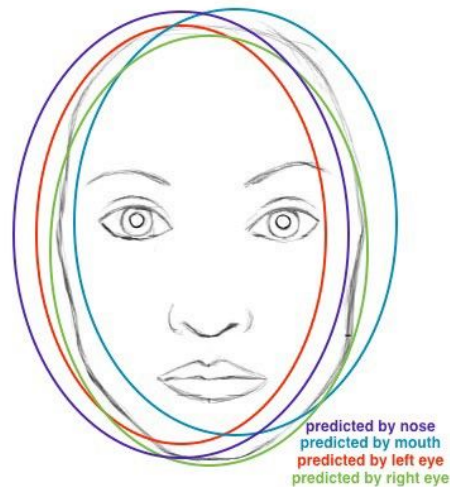
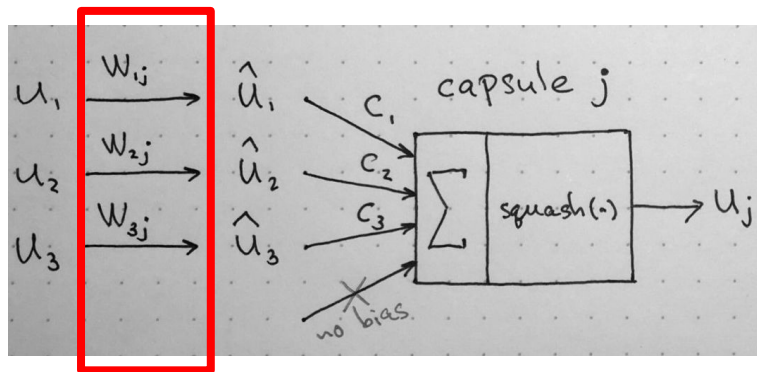
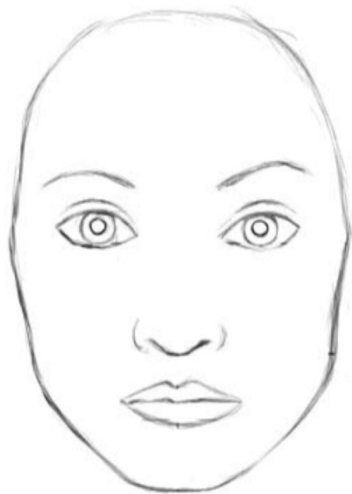


1. Matrix multiplication of input vectors
2. Scalar weighting of input vectors
3. Sum of weighted input vectors
4. Vector->Vector nonlinearity



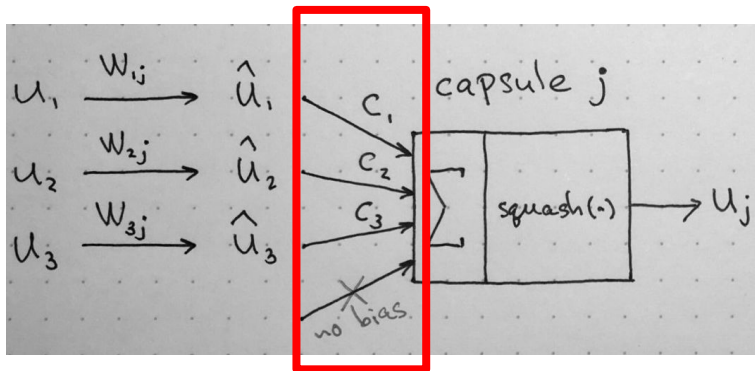
Capsule Calculations 1 - Matrix Multiplication

Lower level feature vectors are weighted by relationship to higher level feature vectors.



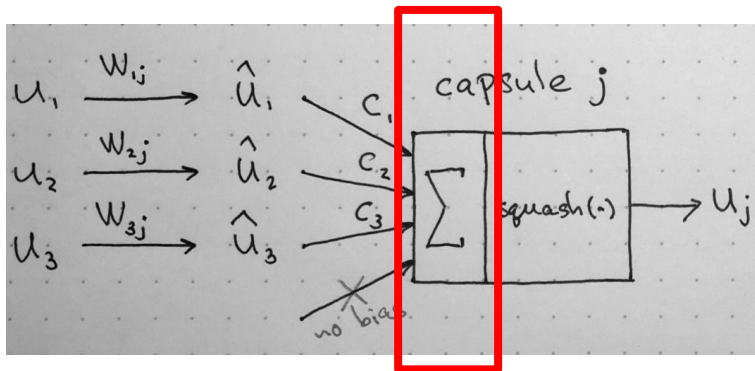
Capsule Calculations 2 - Scalar Weighting of Input Vectors

Weights determined by “dynamic routing” decide the influence of lower capsule on higher capsule.



Capsule Calculations 3 - Sum of Weighted Input Vectors

Similar to regular neuron simulation, except summing vectors rather than scalars.



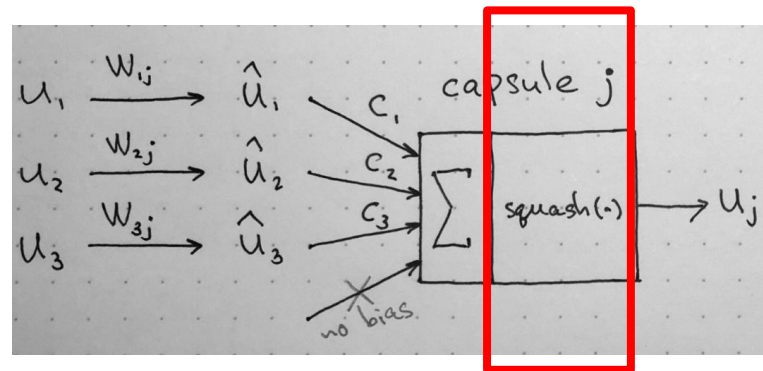
Capsule Calculations 4 - Vector->Vector Nonlinearity

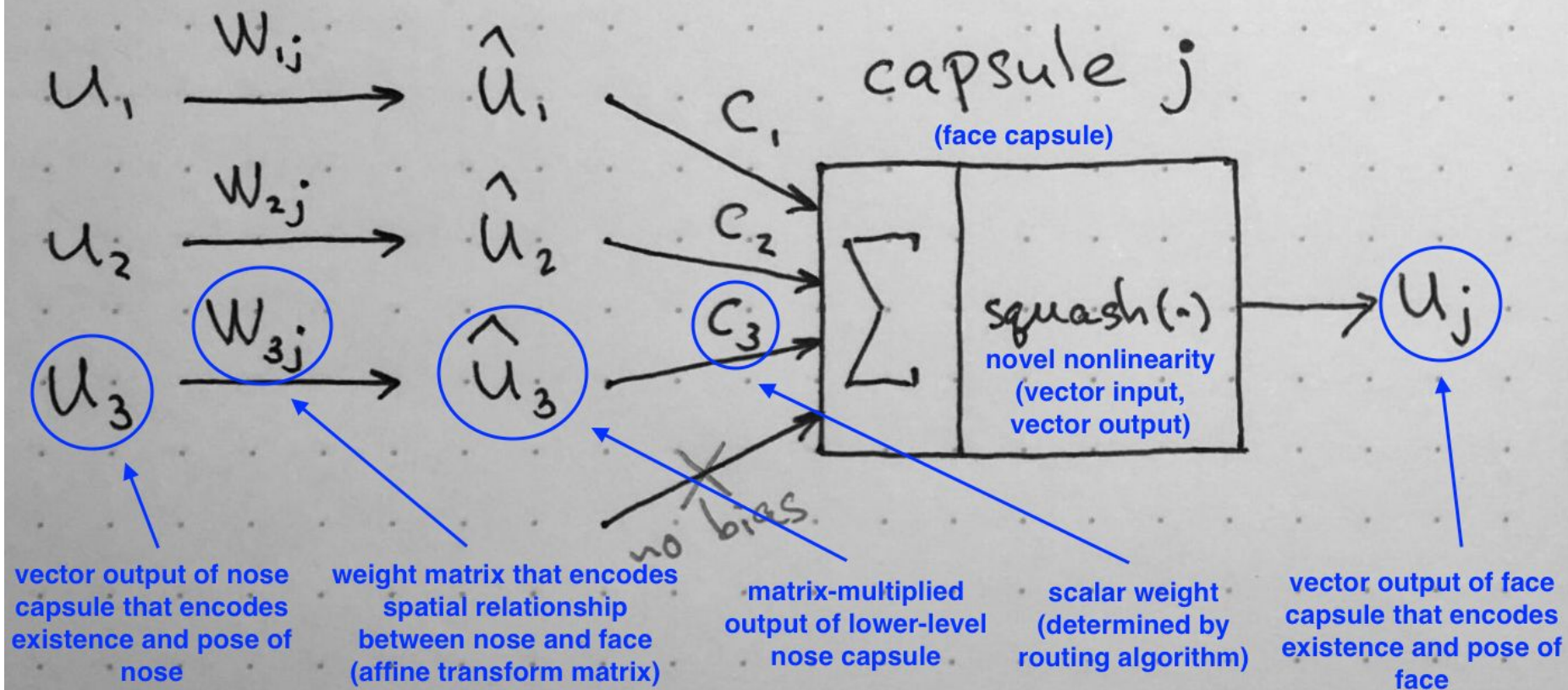
“Squash” function.

Outputs:

- Max length of 1
(unit scaling)
- Maintains
orientation

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

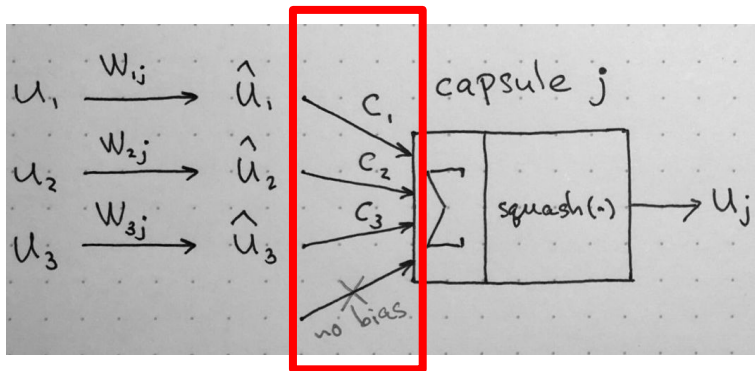




Dynamic Routing

Capsule Calculations 2 - Scalar Weighting of Input Vectors

Weights determined by “dynamic routing” decide the influence of lower capsule on higher capsule.



The Dynamic Routing Algorithm

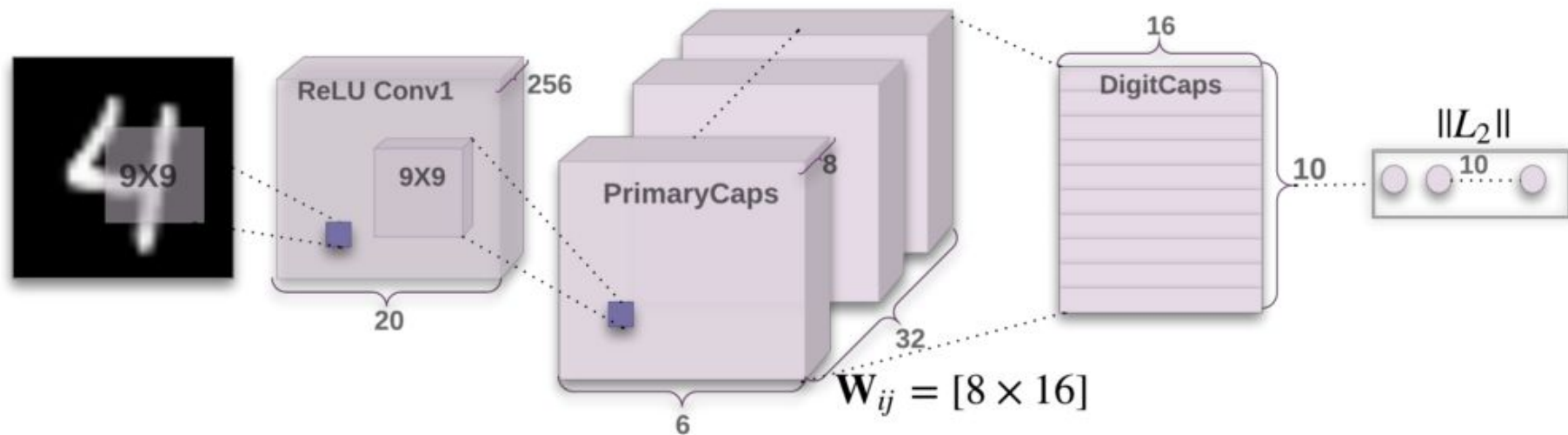
Procedure 1 Routing algorithm.

```
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  $\mathbf{v}_j$ 
```

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

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1 + \|\mathbf{s}_j\|^2} \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$$

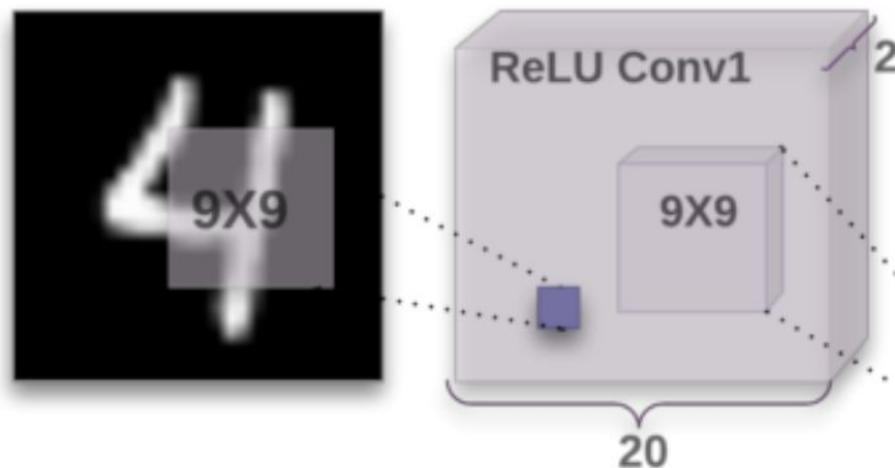
CapsNet Architecture



Convolution 1

- 256, 9×9 convolution kernels
- stride of 1
- ReLU activation.

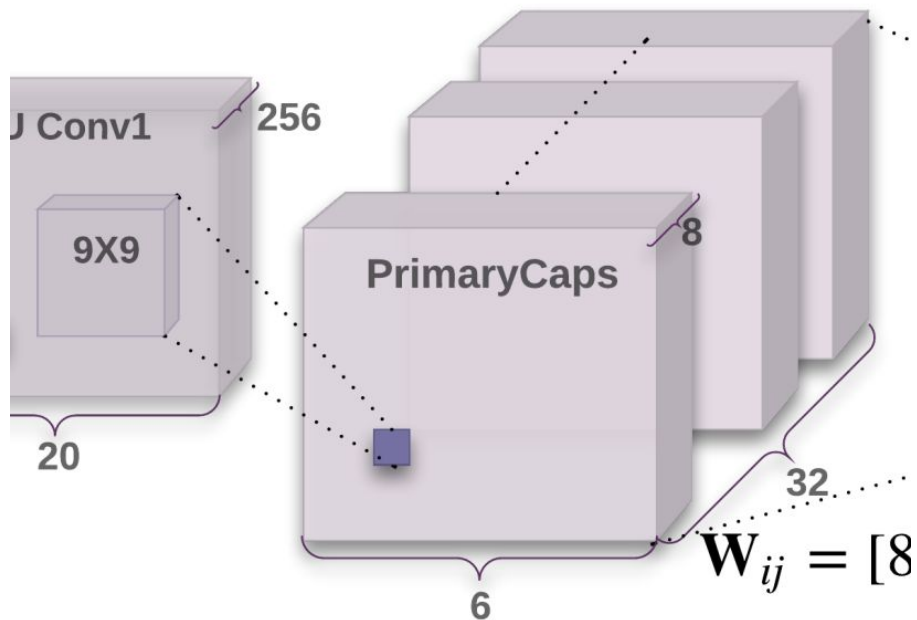
Converts pixel intensity to activities of small feature detectors.



PrimaryCapsules

- Convolutional
- Capsule
- 32 Channels
- 8-Dimensional Channels

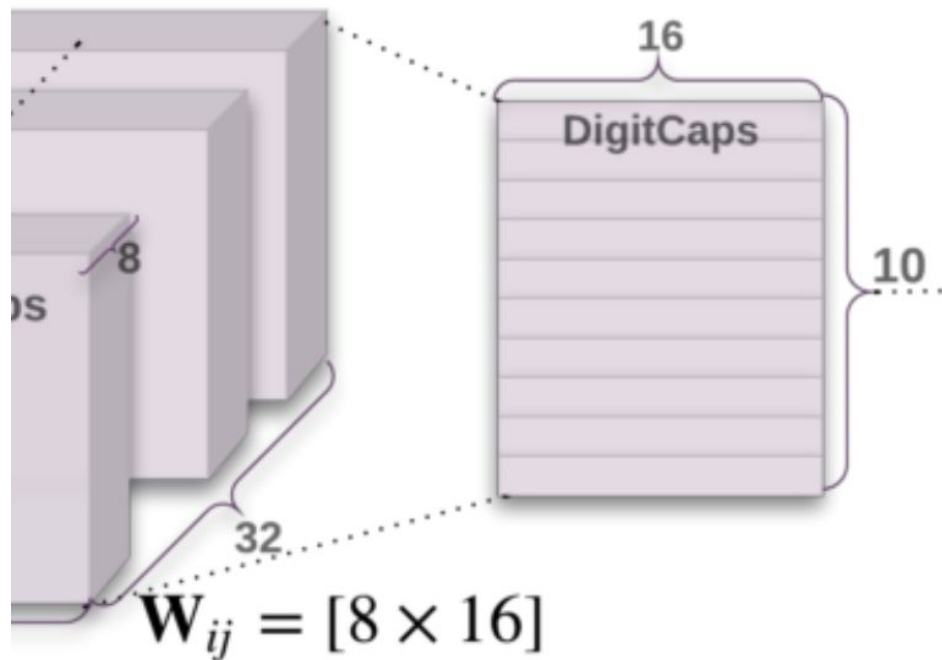
“from an inverse graphics perspective, activating the primary capsules corresponds to inverting the rendering process”



DigitCaps

- 16-Dimensional Capsules
- One capsule per digit classification: total 10

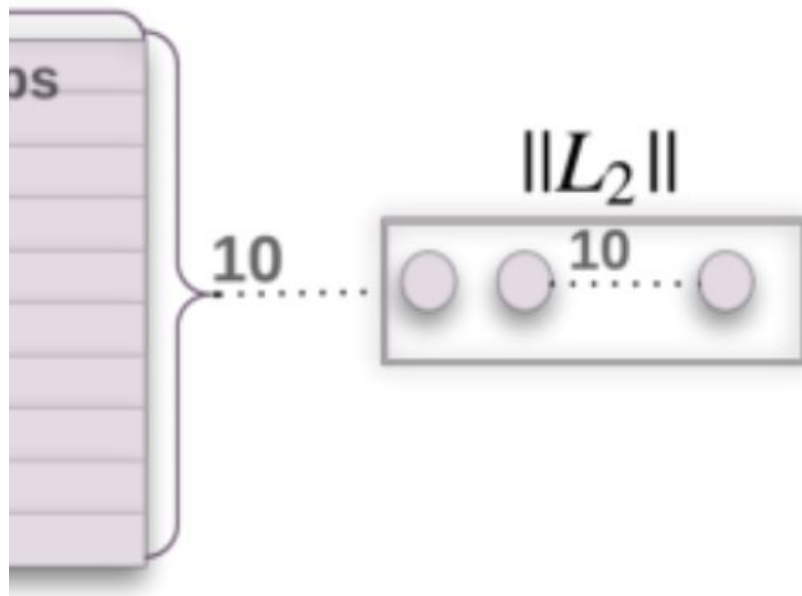
Magnitude of these 16D vectors indicates probability of existence.



Classification Output

- Magnitude of each 16D vector from DigitCaps

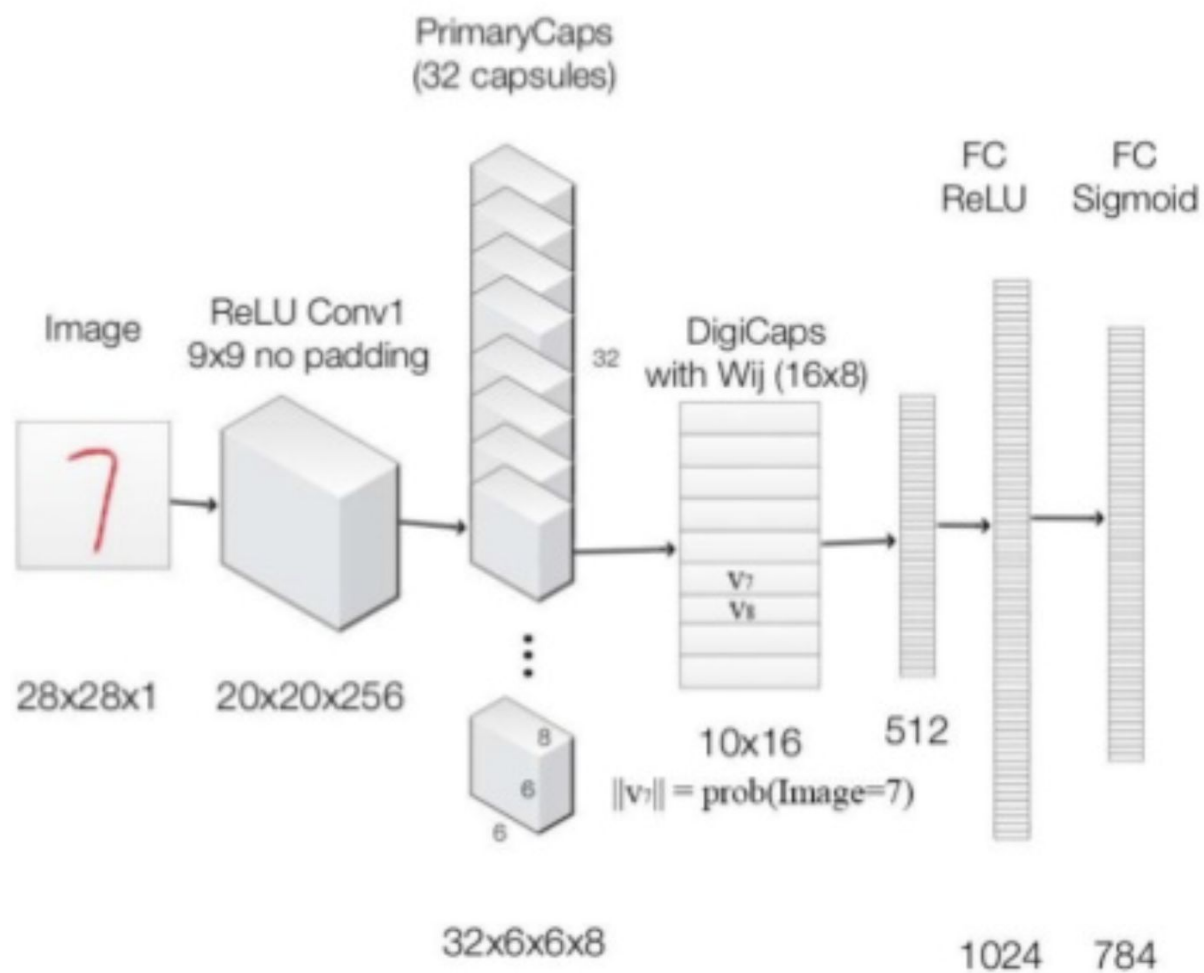
Invariance occurs at this step, losing all pose information



Training Process

— — —

- Tensorflow
- Adam Optimizer - default parameters
- Dynamic Routing used on PrimaryCapsules -> DigitCaps
- Additional loss calculated from MSE of actual digit pixel input and pixels reconstructed from DigitCaps information



Experiments & Results

MNIST

— — —

Dataset:

- 60K training images
- 10K test images
- 28x28 pixel images
- Shifted by up to 2 pixels in each direction with padding

MNIST Results & Comparison

State of the Art:

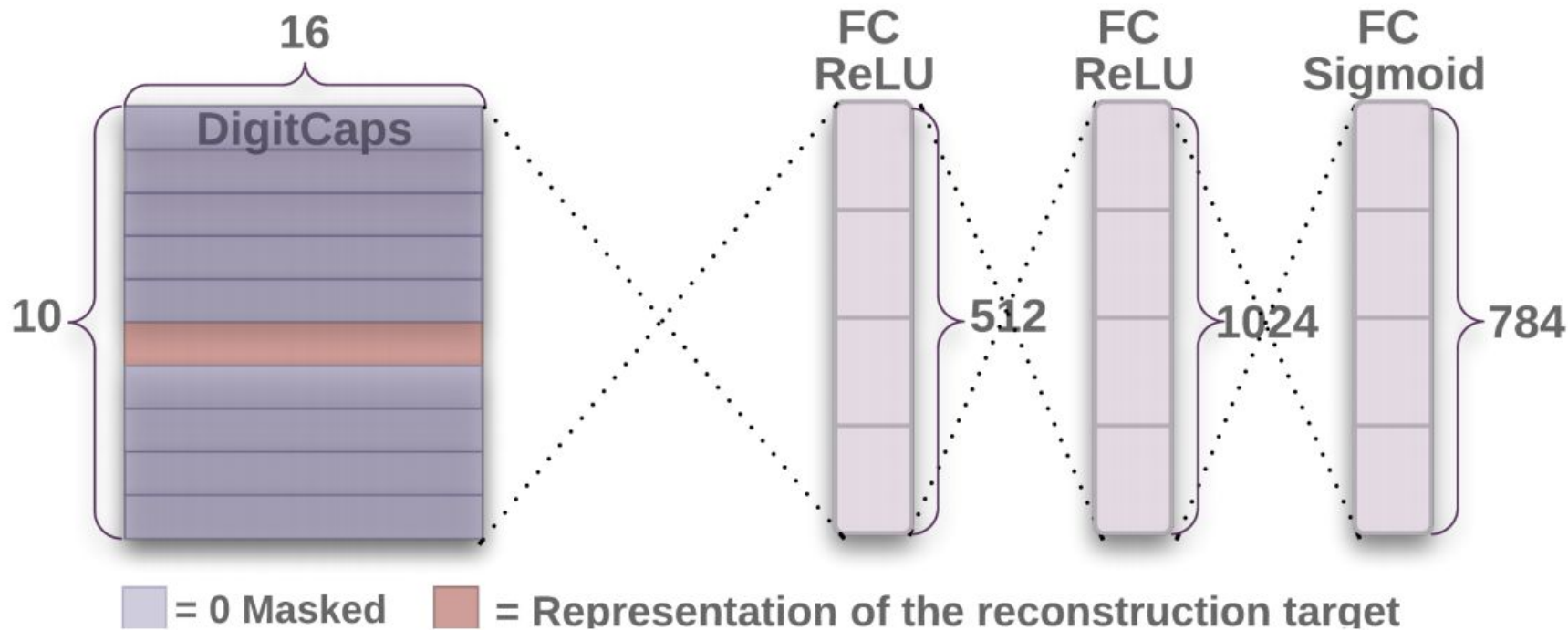
- 0.21 % test error
- Ensembling
- Data augmentation
 - Rotation
 - Scaling
- 0.39 % test error w/o augmentation

CapsNet:

- 0.25 % test error
- 3 layer network







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	0.25 \pm 0.005	5.2

What is in the DigitCaps Layer?



What is in the DigitCaps Layer?

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	

Robustness to Affine Transformations



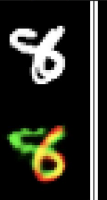
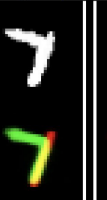









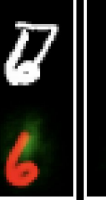


— — —

- Trained with translated MNIST dataset
- Tested on affNIST, w/ random affine transformations
- CapsNet achieved 79% accuracy on affNIST
- Traditional ConvNet achieved 66% accuracy with same training regime

Overlapping Digits

Dataset:

- 80% average overlap
- 60M training images
- 10M test images

R:(2, 7) L:(2, 7)	R:(6, 0) L:(6, 0)	R:(6, 8) L:(6, 8)	R:(7, 1) L:(7, 1)	*R:(5, 7) L:(5, 0)	*R:(2, 3) L:(4, 3)	R:(2, 8) L:(2, 8)	R:P:(2, 7) L:(2, 8)
							
R:(8, 7) L:(8, 7)	R:(9, 4) L:(9, 4)	R:(9, 5) L:(9, 5)	R:(8, 4) L:(8, 4)	*R:(0, 8) L:(1, 8)	*R:(1, 6) L:(7, 6)	R:(4, 9) L:(4, 9)	R:P:(4, 0) L:(4, 9)
							

Discussion

Discussion Points

- CNNs exponential inefficiencies - generalizing to novel viewpoints requires large datasets and deep networks.
- Capsules automatically generalize to new viewpoints
- Capsules are weak to crowding (>1 instance present)
- Research on Capsules is at the same stage as recurrent neural networks was ~2000
- Unparalleled performance on overlapping digits is extremely promising
- CapsNet is learning vector subspaces from the data