

SUMMARY

INTRODUCTION

WHAT IS WRONG WITH CNNs?

CAPSULE NETWORKS

RESULTS

CONCLUSION

INTRODUCTION

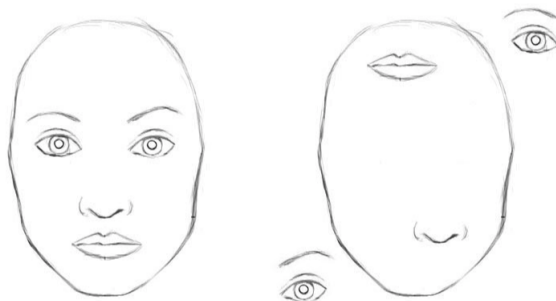
- ▶ **Capsule networks (CapsNets)** are a hot new neural net architecture proposed by Hinton's group
- ▶ It's a promising method that may have a profound impact on deep learning, in particular for computer vision
- ▶ However, before we talk about the capsules, we need to have a look at CNNs

CNN'S DRAWBACKS

- ▶ There is no doubt CNN is a very **important** tool
 - ▶ One of the reasons for the deep learning success
- ▶ But it has some **drawbacks**:
 - ▶ *Backpropagation*:
 - ▶ It needs a huge dataset to be effective
 - ▶ *Translation invariance*:
 - ▶ Objects with orientation or position changes might not be recognized
 - ▶ *Pooling layers*:
 - ▶ It loses a lot of information
 - ▶ It ignores the relation between the part and the whole

CNN'S DRAWBACKS

- ▶ To a CNN, both pictures are similar:



- ▶ The **presence** of the mouth, eyes and nose are a strong indicator to the CNN
- ▶ **Orientational** and **relative spatial** relationships between these components are not very important to the model

CNN'S DRAWBACKS

- ▶ The pooling layer tries to handle this problem
 - ▶ But it's not good enough

- ▶ According to Hinton:

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

- ▶ Thus, it is needed a new approach to handle this issue

INVERSE GRAPHICS APPROACH

- ▶ Inspiration from computer graphics
 - ▶ It deals with constructing a visual image from a **internal representation of geometric data**.
- ▶ This representation is stored in computer's memory
 - ▶ Arrays of geometrical objects
 - ▶ Matrices that represent relative **positions** and **orientation** of these objects
- ▶ Lastly, a special software takes that representation and **converts** it into an image on the screen
 - ▶ This is called **rendering**

INVERSE GRAPHICS APPROACH

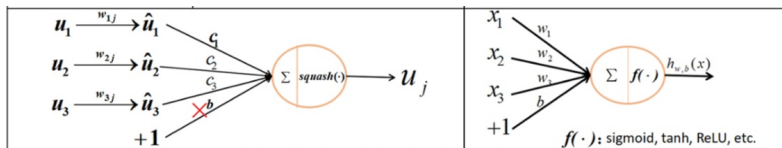
- ▶ Hinton argues that brains do the opposite of rendering:
inverse graphics
 1. Visual information is received by eyes
 2. The eyes deconstruct a hierarchical representation of world
 3. The brain tries to make relationships with already learned patterns
- ▶ Key idea: representation of objects in the brain **does not depend** on view angle
- ▶ How to model it in a neural network?
 - ▶ In 3D graphics, relationships between 3D objects can be represented by a so-called **pose**
 - ▶ Pose is the combination of **position** and **orientation** of an object

INVERSE GRAPHICS APPROACH

- ▶ According Hinton, it is very important to preserve hierarchical pose relationships between object parts
- ▶ This is the key intuition of capsule theory
 - ▶ This is closer to what the human brain does in practice
 - ▶ To recognize digits, the brain needs hundreds of examples, at most
 - ▶ CNNs need thousands
 - ▶ That's why it seems like a brute force approach

CAPSULE NETWORKS

- ▶ A capsule is a new version of a neuron
 - ▶ It encapsulates all important information about the state of the feature they are detecting in **vector form**



- ▶ The probability of detection of a feature is provided by the length of the output vector
- ▶ The state of the detected feature is encoded the direction in which the vector points

CAPSULE NETWORKS

- ▶ When a detected feature moves around the image the vector's length does not change
 - ▶ But its orientation does
- ▶ Hinton calls it by **equivariance**:
 - ▶ The activities changes when a object moves in the picture
 - ▶ The probabilities of detection **remain** constant
 - ▶ The CNN tries to achieve this behavior using max pooling. But it fails.

CAPSULE NETWORKS

		capsule	vs.	traditional neuron
Input from low-level neuron/capsule		vector(u_i)		scalar(x_i)
Operation	Affine Transformation	$\hat{u}_{j i} = W_{ij} u_i$ (Eq. 2)		—
	Weighting	$s_j = \sum_i c_{ij} \hat{u}_{j i}$ (Eq. 2)		$a_j = \sum_{i=1}^3 W_i x_i + b$
	Sum			
	Non-linearity activation fun	$v_j = \frac{\ s_j\ ^2}{1 + \ s_j\ ^2} \frac{s_j}{\ s_j\ }$ (Eq. 1)		$h_{w,b}(x) = f(a_j)$
output		vector(v_i)		scalar(h)

- The capsule may be described in 4 steps:

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

MATRIX MULTIPLICATION OF INPUT VECTOR

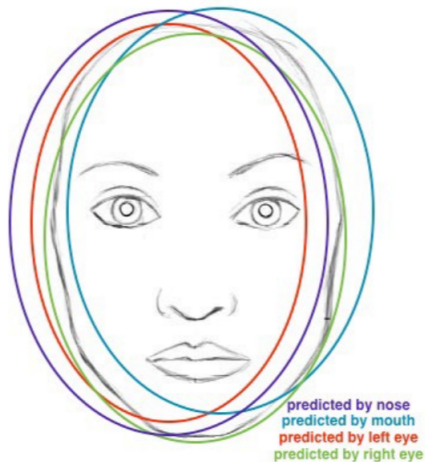
- ▶ Operation: $\hat{\mathbf{u}}_{j|i} = W_{ij}u_i$
- ▶ The input vector $\hat{\mathbf{u}}$ comes from capsules in the layer below
- ▶ Let us consider 3 lower layer capsules \mathbf{u}_1 , \mathbf{u}_2 and \mathbf{u}_3
 - ▶ They detect eye, mouth and nose, respectively
 - ▶ An output capsule \mathbf{v} detects the face
- ▶ These vectors are multiplied by the corresponding \mathbf{W}
 - ▶ It encodes important spatial and other relationships between lower and higher features

MATRIX MULTIPLICATION OF INPUT VECTOR

- ▶ For example: \mathbf{W}_{1j} may encode relationship between nose and face
 - ▶ Face is centered around the nose
 - ▶ Its size is 10 times the noses' one
- ▶ Thus, \mathbf{u}_1 , represents where the face should be according to the nose
- ▶ Similar intuitions can be made to the rest of vectors \mathbf{u}_2 and \mathbf{u}_3
- ▶ If $\hat{\mathbf{u}}_1$, $\hat{\mathbf{u}}_2$ and $\hat{\mathbf{u}}_3$ points to the same direction of \mathbf{v} , it must be a face

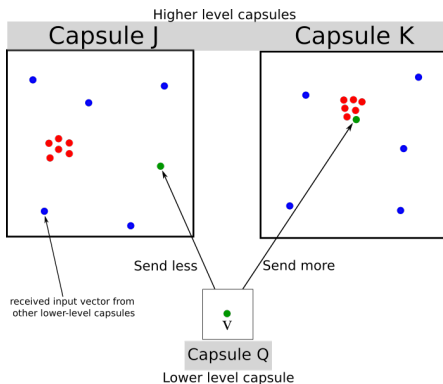
MATRIX MULTIPLICATION OF INPUT VECTOR

- Prediction for the face location:



SCALAR WEIGHTING OF INPUT VECTORS

- ▶ A lower level capsule needs to “decide” to which higher level capsule it will send its output
- ▶ It will make its decision by adjusting the weights C
 - ▶ Dynamic routing based on agreement



SCALAR WEIGHTING OF INPUT VECTORS

- ▶ Which capsule should Q send its output?
 - ▶ This is the essence of the dynamic routing algorithm
- ▶ \mathbf{v} multiplied by \mathbf{W} lands closer to the cluster of red vectors in K
 - ▶ The algorithm must increase the weight \mathbf{C} between the connection $Q-K$
- ▶ The dynamic routing algorithm has a mechanism to achieve this behavior

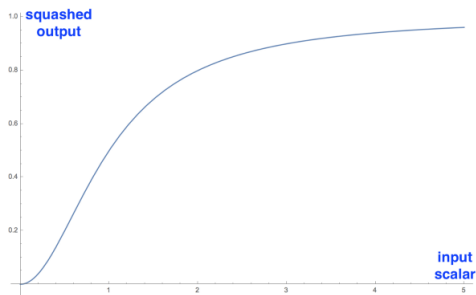
SUM OF WEIGHTED INPUT VECTORS

- ▶ Operation: $\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
- ▶ It works in the same way of the standard neural

VECTOR-TO-VECTOR NONLINEARITY

- CapsNet uses a new nonlinear function called "squash"
 - It squashes the vector to have length between $[0,1]$ without change its direction

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



DYNAMIC ROUTING BETWEEN CAPSULES

*Lower level capsule will send its input to the higher level capsule that “**agrees**” with its input. This is the essence of the dynamic routing algorithm.*

- ▶ The goal of this algorithm is to determine the weights c_{ij}
 - ▶ It's a non-negative scalar
 - ▶ For each lower layer capsules, the sum of all weights equals 1
- ▶ The characteristics of c_{ij} allow us to interpret them in probabilistic terms
 - ▶ This defines a probability distribution of its output belonging to each higher level capsule

DYNAMIC ROUTING BETWEEN CAPSULES

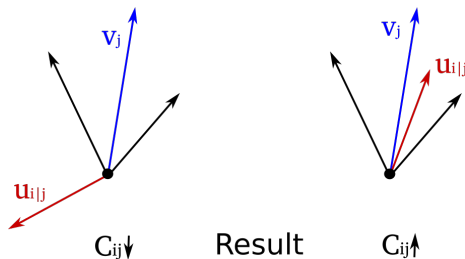
Procedure 1 Routing algorithm.

```
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  $l$ :  $\mathbf{c}_i \leftarrow \text{softmax}(\mathbf{b}_i)$ 
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  $(l + 1)$ :  $\mathbf{v}_j \leftarrow \text{squash}(\mathbf{s}_j)$ 
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  $\mathbf{v}_j$ 
```

- ▶ It takes all outputs $\hat{\mathbf{u}}$ from all capsules in a lower level l
- ▶ It will provide the output \mathbf{v}_j for a higher layer

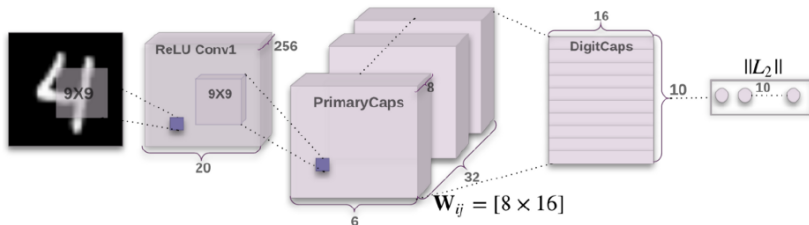
DYNAMIC ROUTING BETWEEN CAPSULES

- ▶ At the beginning: $b_i = 0 \rightarrow c_i = 0.5$
 - ▶ State of maximum **confusion** and **uncertainty**
 - ▶ The lower capsule **have no idea** which higher capsule will best fit their output
- ▶ At the last step: $b_{ij} = b_{ij} + \hat{\mathbf{u}}_{j|i} \times \mathbf{v}_j$
 - ▶ This step ensures the "agreement"
 - ▶ The dot product can be understood as a measure of similarity



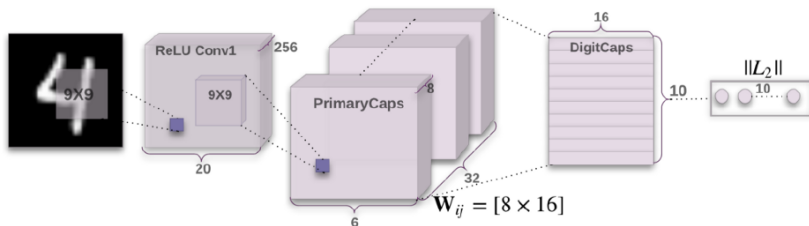
CAPSNET ARCHITECTURE

- CapsNet architecture for MNIST dataset - Encoder:



- This architecture has 3 layers: 2 convs and 1 caps
- It learns to encode it into a 16-dimensional vector of instantiation parameters

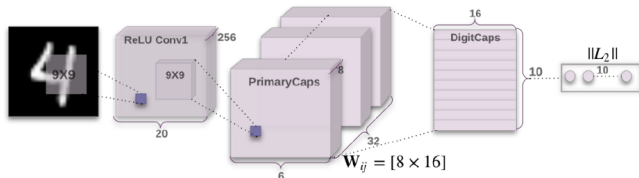
CAPSNET ARCHITECTURE



► Conv1:

- 256 filters 9×9 , stride = 1, no padding, relu activation
- *input*: $28 \times 28 \times 1$ images
- *output*: $20 \times 20 \times 256$

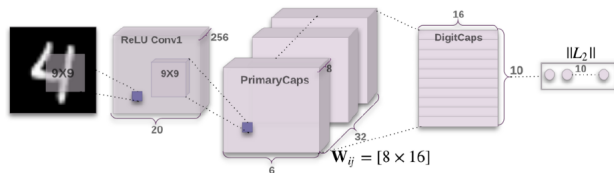
CAPSNET ARCHITECTURE



► PrimaryCaps:

- It aims to take basic features detected by the conv1 and produce combinations of them
- 32 primaryCaps
 - Very similar to convLayer
 - Each caps applies **eight** 256 filters 9×9 to the conv1 output
 - As the stride = 2, it produces 32 blocks of $6 \times 6 \times 8$
 - We can think the result as 1152 ($6 \times 6 \times 32$) 8D vectors

CAPSNET ARCHITECTURE



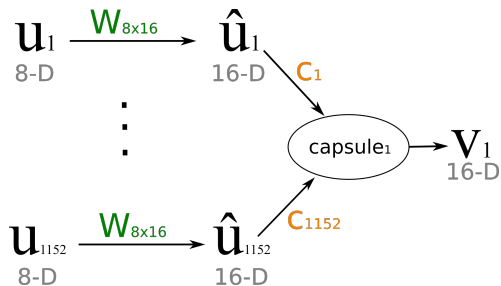
► DigitCaps:

- It has 10 caps, one for each digit, of 16 dimension
- Each caps takes as input a $6 \times 6 \times 32$ 8-dimensional vector (1152 8-D vectors)
- Each input vectors gets their own 8×16 weight matrix W
 - W maps 8-D to 16-D

CAPSNET ARCHITECTURE

► DigitCaps

- For the first capsule:



CAPSNET ARCHITECTURE

► Loss Function

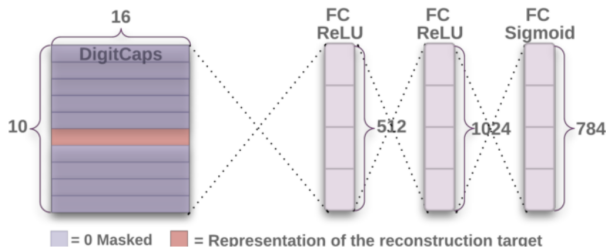
- The model uses a special loss function:

$$L_k = T_k \max(0, m^+ - \|\mathbf{v}_k\|)^2 + \lambda(1 - T_k) \max(0, \|\mathbf{v}_k\| - m^-)^2 \quad (2)$$

- $T_k = 1$ if the digit of class k is present, or 0 otherwise.
- $m^+ = 0.9$, $m^- = 0.1$ and $\lambda = 0.5$.

CAPSNET ARCHITECTURE

► Decoder



- It takes a 16-D vector from the **correct** DigitCaps and learns to reconstruct it into an digit image
- Reconstruction loss: Euclidean distance
- Final loss: $L = LF + \alpha RL, \alpha = 0.0005$

EXPERIMENTAL RESULTS

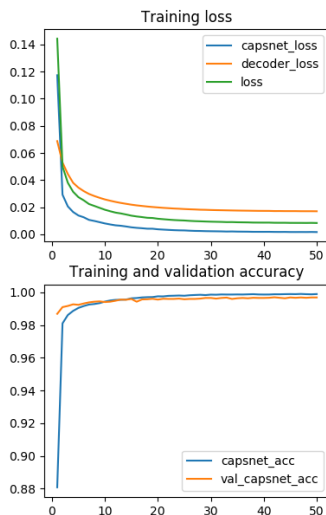
Model	Routing	Decoder	Error rate
CapsNet1	1	NO	0.39 ± 0.024
CapsNet1	1	YES	0.36 ± 0.009
CapsNet1	3	NO	0.40 ± 0.016
CapsNet1	3	YES	0.34 ± 0.016
[1]	3	YES	0.25 ± 0.005

Table: Error rate for MNIST dataset

- ▶ Batch size = 50, Epochs = 50, Optimizer: Adam, lr: 0.001
- ▶ Each epoch takes around 140 sec

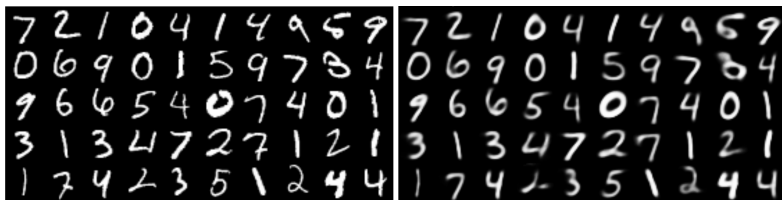
EXPERIMENTAL RESULTS

► Convergence graphs



EXPERIMENTAL RESULTS

► Reconstruction:















Original

Reconstructed

EXPERIMENTAL RESULTS

► Reconstruction [1]:

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

CONCLUSION

- ▶ This is promising for image segmentation and object detection
- ▶ It Requires less training data
- ▶ It tries to follow the brain algorithm
- ▶ It needs to be tested in larger datasets (ImageNet)
- ▶ Due to the inner loop in the routing, it is a bit slow
- ▶ It is proposed by Hinton, thus, we need to pay attention

REFERENCES

1. Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in Neural Information Processing Systems. 2017.
2. Pechyonkin, Max. "Understanding Hinton's Capsule Networks", Available on: <https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b>. 2017.
3. Hinton, Geoffrey, Nicholas Frosst, and Sara Sabour. "Matrix capsules with EM routing." 2018.
4. Implementations:
 - ▶ <https://github.com/XifengGuo/CapsNet-Keras>
 - ▶ https://github.com/ageron/handson-ml/blob/master/extra_capsnets.ipynb