Dynamic Routing Between Capsules

Authors: Sara Sabour, Nicholas Frosst, Geoffrey E Hinton

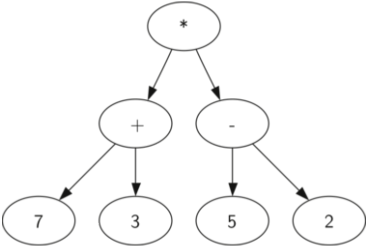
November 2017

Introduction

- A capsule is a group of neurons whose activity vector represents an entity - either an object or an object part
- The length of the vector represents the probability that a certain entity exists
- Achieves state-of-the-art performance on MNIST and is better at recognizing highly overlapping digits than a convolutional network

Reference with human vision

- The human vision processes only a tiny fraction of the optic array at the highest resolution
- Assumption that the visual system creates a parse tree (Fig. a) on each fixation



(a) Parse tree example:
(7 + 3) * (5 - 2)

Capsules as parse trees

- A parse tree can be represented by a multilayer neural network
- Each layer will be divided into many capsules
- Using an iterative routing process, each capsule will choose a capsule in the layer above to be its parent in the tree - this process solves the problem of assigning parts to wholes

Describing the capsule

- The vector of activities of a capsule represent the various properties of a particular entity that is present in the image (pose, deformation, velocity, texture etc.)
- The existence of an object is defined by using the overall length of the activity vector of a capsule - a non-linearity is applied so that the length of the vector output cannot exceed 1
- A lower level capsule is assigned to a higher level capsule by a "routing-by-agreement" algorithm

How the vector inputs and outputs of a capsule are computed

The length of the output vector of a capsule should represent the probability that the entity represented by the capsule is present in the current input. A non-linear squashing function is used to ensure that short vectors get shrunk to almost zero length and long vectors get shrunk to a length slightly below 0:

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|^2} \tag{1}$$

where v_j is the vector output of capsule j and s_j is its total input.

How the vector inputs and outputs of a capsule are computed

For all but the first layer of capsules, the total input to a capsule s_j is a weighted sum over all prediction vectors $\hat{u}_{j|i}$ from the capsules in the layer below and is produced by multiplying the output u_i of a capsule in the layer below by a weight matrix W_{ij}:

$$s_j = \sum_i c_{ij} \hat{u}_{i|j}, \quad \hat{u}_{i|j} = W_{ij} u_i \tag{2}$$

where the c_{ij} are coupling coefficients that are determined by the iterative dynamic routing process.

How the vector inputs and outputs of a capsule are computed

The coupling coefficients between capsule i and all the capsules in the layer above sum to 1 and are determined by a "routing softmax" whose initial logits b_{ij} are the log prior probabilities that capsule i should be coupled to capsule j:

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{3}$$

The initial coupling coefficients are iteratively refined by measuring the agreement between the current output v_j of each capsule, j, in the layer above and the prediction $\hat{u}_{j|i}$ made by capsule i.

How the vector inputs and outputs of a capsule are computed

The agreement is simply the scalar product $a_{ij} = v_j \cdot \hat{u}_{j|i}$. This agreement is treated as if it were a log likelihood and is added to the initial logit, b_{ij} before computing the new values for all the coupling coefficients linking capsule i to higher level capsules.

Routing algorithm

function $\mathrm{ROUTING}(\hat{u}_{j|i}, r, l)$

for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.

for r iterations **do**

for all capsule i in layer l: $c_i \leftarrow \mathrm{softmax}(b_i)$

for all capsule j in layer $(l+1)$: $s_j \leftarrow \sum_i c_{ij}\hat{u}_{j|i}$

for all capsule j in layer $(l+1)$: $v_j \leftarrow \mathrm{squash}(s_j)$

for all capsule i in layer l and capsule j in layer $(l+1)$:

$b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$

return v_j

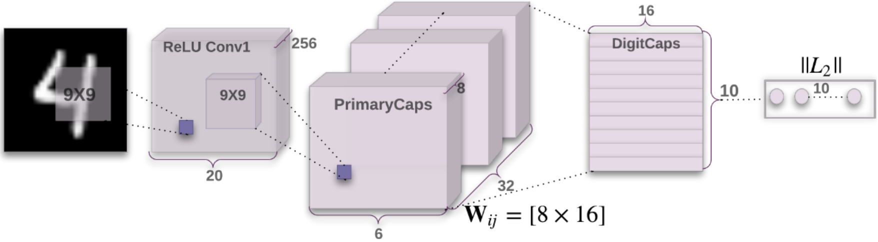
Margin loss for digit existence

Because the length of the instantiation vector is used to represent the probability that a capsule's entity exists, the top-level capsule for digit class k should have a long vector if and only if that digit is present in the image. For multiple digits, a margin loss is used:

$$L_c = T_c max(0, m^+ - \|v_c\|)^2 + \lambda(1 - T_c) max(0, |v_c\| - m^-)^2 \quad (4)$$

where $T_c = 1$ iff a digit of class c is present and $m^+ = 0.9$ and $m^- = 0.1$. The λ 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. The total loss is simply the sum of the losses of all digit capsules.

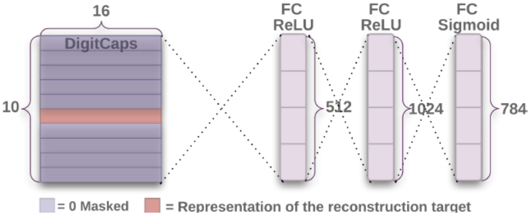
CapsNet architecture



(b) CapsNet architecture

- Shallow network with only two convolutional layers and one fully connected layer (Fig. b)
- Routing is implemented only between PrimaryCaps and DigitCaps layers

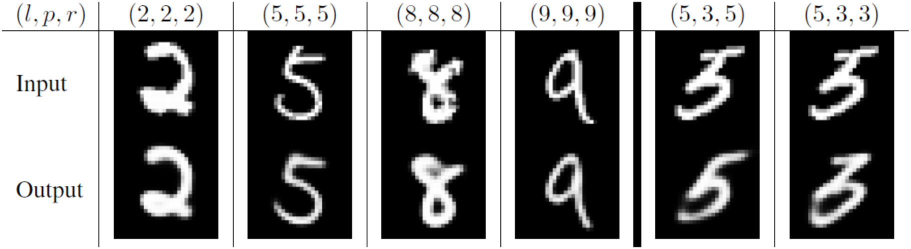
Reconstruction as a regularization method



(c) Reconstruction from a capsule representation

An additional reconstruction loss is used to encourage the digit capsules to encode the instantiation parameters of the input digit. The output of the correct digit capsule is fed into a decoder consisting of 3 fully connected layers that model the pixel intensities (Fig. c). The objective function is the sum of squared differences between the outputs of the logistic units and the pixel intensities. This loss is scaled by 0.0005 so that it does not dominate the margin loss during training.

Reconstruction results



(d) Digits reconstructed from the output of a capsule

Results on MNIST

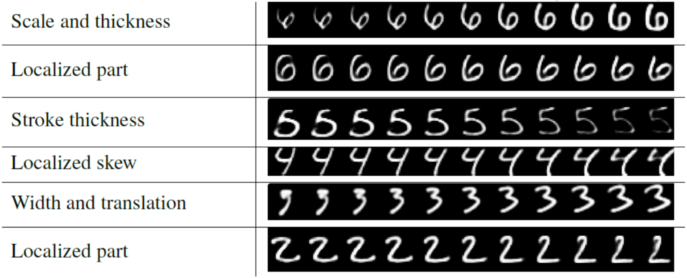
Method	Routing	Reconstruction	MNIST (%)	MultiMNIST (%)
Baseline	-	-	0.39	8
CapsNet	1	no	$0.34_{\pm 0.032}$	-
CapsNet	1	yes	$0.29_{\pm 0.011}$	7
CapsNet	3	no	$0.35_{\pm 0.036}$	-
CapsNet	3	yes	$\mathbf{0.25}_{\pm 0.005}$	**5**

(e) Comparations between different network configurations and a CNN baseline

The baseline is a standard CNN with three convolutional layers of 256, 256, 128 channels. Each has 5x5 kernels and stride of 1. The last convolutional layer is followed by two fully connected layers of size 328, 192. The last fully connected layers is connected with dropout to a 10 class softmax layer with cross entropy loss.

What the individual dimensions of a capsule represent

- The dimensions of a digit capsule should learn to span the space of variations in the way digits of that class are instantiated (stroke thickness, skew and width)
- To see what the individual dimensions represent, a perturbed output vector can be fed to the decoder network to see how the perturbation affects the reconstruction (Fig. f)



(f) Dimension peturbations

Robustness to Affine Transformations

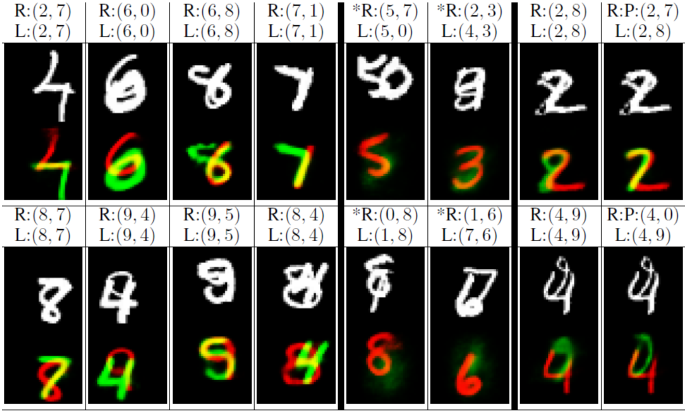
- Experiments show that each DigitCaps capsule learns a more robust representation for each class than a traditional convolutioanl network
- To test the robustness of CapsNet to affine transformations, a training was done on a padded and translated MNIST set, in which each example is an MNIST digit placed randomly on a black background
- The network was tested on the affNIST[1] data set, in which each example is an MNIST digit with a random small affine transformation
- CapsNet achieved 79% accuracy, while a tranditional convolutional model with a similar number of parameters only 66%

[1] Available at `http://www.cs.toronto.edu/~tijmen/affNIST/`.

Segmenting highly overlapping digits

- Dynamic routing can be viewed as a parallel attention mechanism that allows each capsule at one level to attend to some active capsules at the level below and to ignore others; This should allow the model to recognize multiple object in the image even if objects overlap
- To test this assumption, the MultiMNIST dataset was created by overlaying a digit on top of another from the same set, but different class, having 80% average overlap
- During reconstruction, a digit is picked one at a time and use the activity vector of the chosen digit capsule to reconstruct the image of the chosen digit (Fig. g)
- The fact that it is able to reconstruct digits regardless of the overlap shows that each digit capsule can pick up the style and position from the votes it is receiving from PrimaryCapsules layer

Segmenting highly overlapping digits



(g) Sample reconstruction of a CapsNet on MultiMNIST test dataset

Other datasets

- On CIFAR10, the network achieved 10.6% error with an ensemble of 7 models; 10.6% test error is about what standard convolutional nets achieved when they were first applied to CIFAR10[2]
- The network was also tested on smallNORB[3] and achieved 2.7% test error rate, which is in-par with the state-of-the-art[4]

[2] Matthew D. Zeiler and Rob Fergus. "Stochastic Pooling for Regularization of Deep Convolutional Neural Networks". In: *CoRR* abs/1301.3557 (2013). arXiv: 1301.3557. URL: http://arxiv.org/abs/1301.3557.

[3] Yann LeCun, Fu Jie Huang, and Léon Bottou. "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. CVPR'04. Washington, D.C., USA: IEEE Computer Society, 2004, pp. 97–104. URL: http://dl.acm.org/citation.cfm?id=1896300.1896315.

[4] Dan C. Ciresan et al. "High-Performance Neural Networks for Visual Object Classification". In: *CoRR* abs/1102.0183 (2011). arXiv: 1102.0183. URL: http://arxiv.org/abs/1102.0183.

Conclusion

- Convolutional neural networks suffer from the difficulty of generalizing to novel viewpoints. Capsules avoid this issue by converting pixel intensities into vectors of instantiation parameters of recognized fragments and then applying transformation matrices to the fragments to predict the instantiation parameters of larger fragments. Transformation matrices that learn to encode the intrinsic spatial relationship between a part and a whole constitute viewpoint invariant knowledge that automatically generalizes to novel viewpoints.
- Capsules can deal with multiple different affine transformations of different objects or object parts at the same time; they are also very good for dealing with segmentation.
- Research on capsules is now at a similar stage to research on recurrent neural networks for speech recognition at the beginning of this century.

Public implementations

- Tensorflow: https://github.com/naturomics/CapsNet-Tensorflow
- Pytorch: https://github.com/gram-ai/capsule-networks
- Keras: https://github.com/XifengGuo/CapsNet-Keras

Further research

- ICLR 2018 blind submission - Matrix Capsules With EM Routing [5]
  - Uses a matrix instead of a vector as the instantiation parameters of a capsule
  - Uses the EM algorithm to iteratively update the coupling coefficients

[5] https://openreview.net/pdf?id=HJWLfGWRb

Technical slides - How PrimaryCaps is computed?

There are two ways to obtain a convolutional capsule from the output of a traditional convolution:

- Perform 8 separate traditional convolutions (each with 32 filters), reshape the output of each of these convolutions to a tensor of $(32 * 6 * 6)x1$, and then concatenate all those tensors along the last axis to obtain a tensor of dimension $(32 * 6 * 6)x8$; the interpretation of this output is that there are $32 * 6 * 6$ capsules, each being an $8D$ vector.
- Perform a single convolution with $32 * 8$ filters and then reshape the output into a tensor of dimension $(32 * 6 * 6)x8$

Technical slides - When is the routing procedure applied?

- The routing procedure is applied when computing the feedforward activities through the network and only between the PrimaryCaps and DigitCaps layers. The coupling coefficients are determined by using the routing mechanism, for the backpropagation phase remaining only the transformation matrices W_{ij} to be computed. The training is done in a classical way, by using the Adam optimizer, the routing module being fully differentiable.