CapsNet Reading Notes

# Capsules vs. Traditional Neurons

|  |  |  |  |
| --- | --- | --- | --- |
| Input from a low-level  capsule / neuron | | (vector) | (scalar) |
| Operations | Affine Transform (仿射变换) |  | None |
| Weighting |  |  |
| Non-linearity |  |  |
| Output | | Vector | Scalar |

Table Comparison between a capsule and a traditional neuron.

# 4 Steps Happening Inside a Capsule (Intuition)

1. Matrix multiplication (affine transformation) of input vector.
2. Scalar weighting of input vectors.
3. Sum of weighted input vectors.
4. “Squashing” non-linearity.



Figure Capsule architecture: how a capsule manipulates input features.

## Affine Transformation

**Weight matrix**

It encodes important spatial (and perhaps other) relationships between lower level features {eyes, mouth, nose} and higher level features {face}.



Figure The significance of the weight matrices in a capsule.

**e.g.**  encodes where the face should be according to the detected position of eyes.

## Scalar Weighting

**Essence**: decide how much of the lower-level output is to send to each of the upper-level capsules. The higher level capsule measures which lower-level capsule better accommodates its result, say Capsule . Then it will automatically adjust its weight so that  increases and other weight coefficients decreases. This mechanism is called Dynamic Routing.

## Summation

Too easy. Omit.

## Squashing

Make the output vector:

* Length: probability (belief) of the existence of a feature.
* Direction: the internal state (e.g. the pose) of the feature.

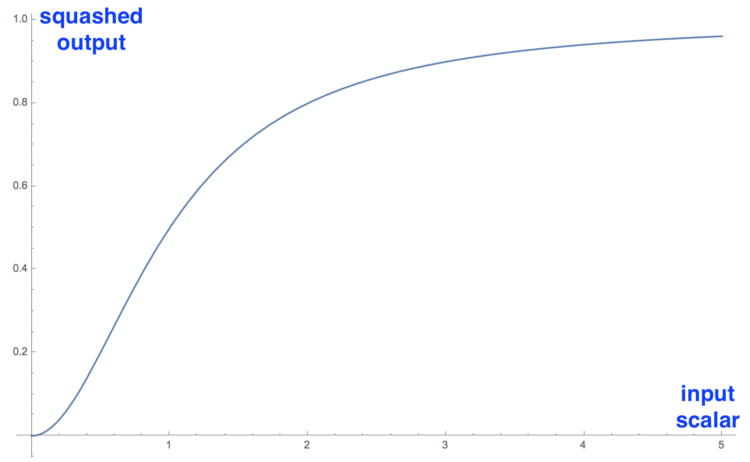


Figure The curve of the magnitude of the squashed output over the magnitude of the input vector.

## Summary

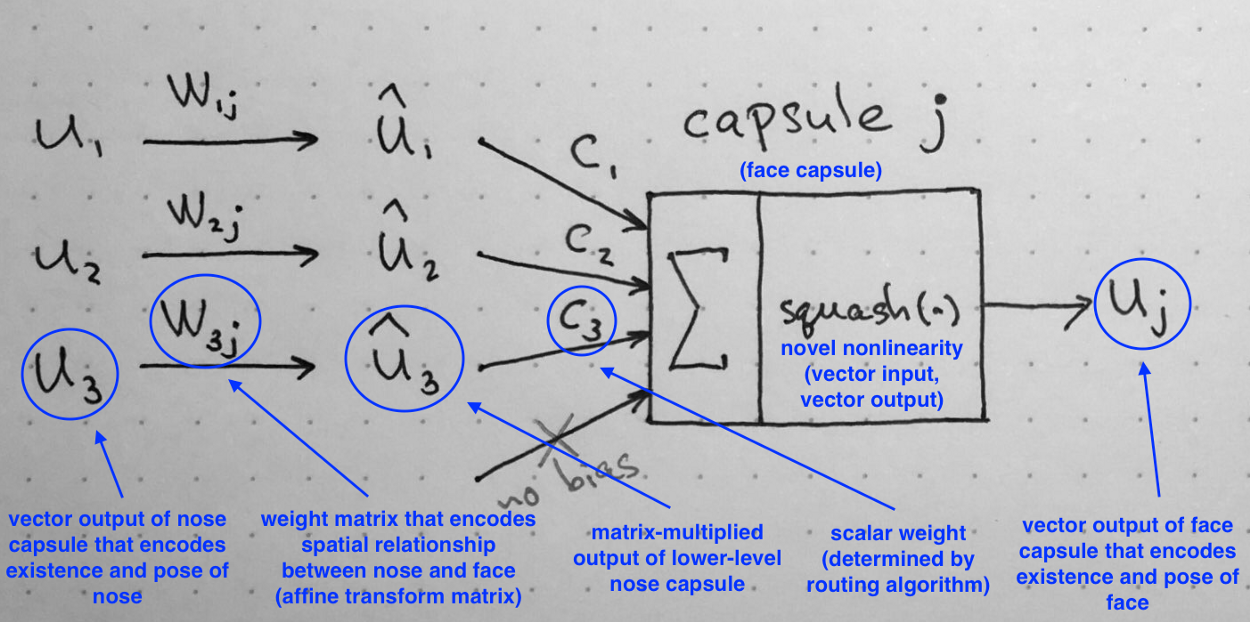


Figure Intuition of all math terms of a capsule.

# More details on Dynamic Routing

## The weight/coefficient of each connection (Intuition)

As is shown in Part II, a capsule in a lower-level layer needs to decide how to send its output vector to higher-level capsules . It makes this decision by changing the scalar weight that multiplies over the lower-level capsule and goes as input to a higher-level capsule .

NOTE: I prefer to call a scalar weight as a coefficient, and this name will be used from here.

## Properties of

1. Each coefficient is a non-negative scalar
2. For each lower-level capsule , the sum of all routing coefficients equals to 1.
3. For each lower-level capsule , the # of coefficients equals to the # of higher-level capsules. (问题: 有点像全连接?)
4. Coefficients are updated by the iterative dynamic routing algorithm.

The first two properties allow us to interpret weights in probabilistic terms. In a sense, for each lower level capsule i, its weights define a **probability distribution** of its output belonging to each higher level capsule .

## What exactly happens during dynamic routing?

### KEEP IN MIND the essence of dynamic routing

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.

\* “its” refers to the lower level caps.

### Go through the algorithm line by line

|  |  |
| --- | --- |
| **Procedure 1** Routing algorithm | |
| 1 | Procedure Routing(, , ): |
| 2 | for capsule in layer : |
| 3 | for capsule in layer : |
| 4 |  |
| 5 | for iterations: |
| 6 | for capsule in layer : |
| 7 |  |
| 8 | for capsule in layer : |
| 9 |  |
| 10 |  |
| 11 | for capsule in layer : |
| 12 | for capsule in layer : |
| 13 |  |
| 14 | return |

Figure The pseudo-code of the Dynamic Routing algorithm.

**Line 1:** Parameters of the procedure.

* : the layer index.
* : the outputs of all capsules in layer .
  + is iterated by .
  + : the output of capsule sending to capsule .
* : the # of iterations.

**Line 2~4:** is simply an intermediate value that will be iteratively updated. After the procedure is over, its value will be stored in. At start of training the value of is initialized at zero.

**Line 5:** the steps in line 6~13 will repeat times.

**Line 6~7:** is the vector containing all routing coefficients for   
capsule . “softmax” means:

Why softmax? Essentially, this ensures that the routing coefficients is a valid probability and their sum is 1.

NOTE: At first, all equal to 0. Given 3 capsules in thus all equal to 0.5. The state of all coefficients being equal represents the state of maximum confusion and uncertainty: a lower level capsules have no idea which higher level capsule will best fit their output.



Figure The state that all routing coefficients being equal implies a complete uncertainty.

**Line 8~10:**

Intuition: these two steps performs “forward propagation”.

Recall that “squash” means:



### Updating routing coefficients: line 11~13

**What does “agreement” mean?**

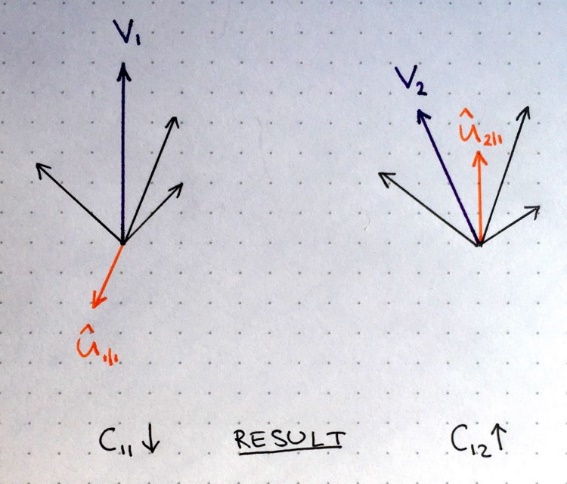
Imagine that there are two higher level capsules:

Figure 7 The routing coefficients are updated by “agreement”.

* Output: purple vectors and .
* Input:
  + orange vectors and   are from lower capsule#1.
  + Other input vectors are black.

In the left part, and the   point to the opposite directions

* They are not similar.
* Their dot product will be a negative number
* The routing coefficient will decrease.

In the right part, and point to the similar direction.

* They are similar.
* Their dot product will be a positive number
* The routing coefficient will increase.

This procedure is repeated for all higher level capsules and for all inputs of each capsule. The result of this is a set of routing coefficients that best matches outputs from lower level capsules with outputs of higher level capsules.

# CapsNet on MNIST

## Encoder



Figure The block diagram of the encoder network.

The encoder is essentially a classifier that decide which digit is the input picture.

### Layer 1. Convolutional layer

|  |  |  |  |
| --- | --- | --- | --- |
| Job | Detect basic features in the 2D image. | | |
| Input | Height | Width | #Channels |
| 28 | 28 | 1 |
| Output | 20 | 20 | 256 |
| Conv Kernels |  |  |  |
| #Parameters | Note that each kernel has a bias term. | | |

### Layer 2. PrimaryCaps Layer



Figure A PrimaryCaps is essentially a 3D convolutional unit activated by the squash function.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Job | Combine the features from layer 1. | | | | |
| Input | Height | Width | | Length/Depth | #Channels |
| 20 | 20 | | 1 | 256 |
| Output | 6 | 6 | | 8  Each capsule outputs an 8D vector. | #of capsules. |
| Convolutional Part |  |  | |  |  |
| #Parameters | |  | | |
| #Parameters |  | | | | |

### Layer 3. DigitCaps Layer



Figure A capsule in DigitCaps Layer is equivalent to the one in Figure 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Job | This layer has 10 digit capsules, one for each digit. | | | |
| Input | Height | Width | Length/Depth | #Channels |
| 6 | 6 | 8 | 32 |
| Output | 1 | 1 | 16 | 10 |
| #Parameters | 1152 input vectors. ∴ 1152 transformation matrices, each of which is 8\*16.  1152 coefficients. 1152 coefficients.  ∴ Each capsule: 1152\*8\*16+1152+1152 = 149760  ∴ All capsules: 1497600 | | | |

### Classification Loss

NOTE: A correct digit cap is one that matches the training label.

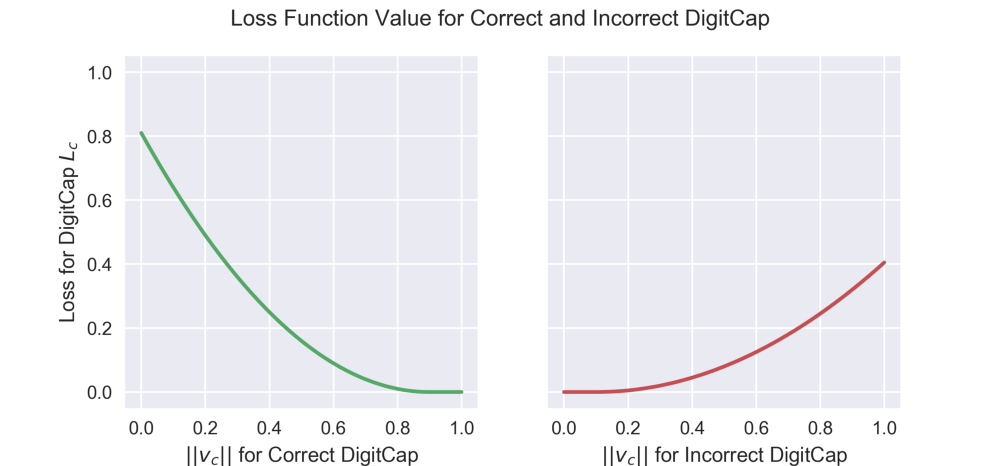


Figure Loss function value for correct and incorrect DigitCap. Note that the red graph is “squashed” vertically compared to the green one. This is due to the lambda multiplier from the formula.

## Decoder

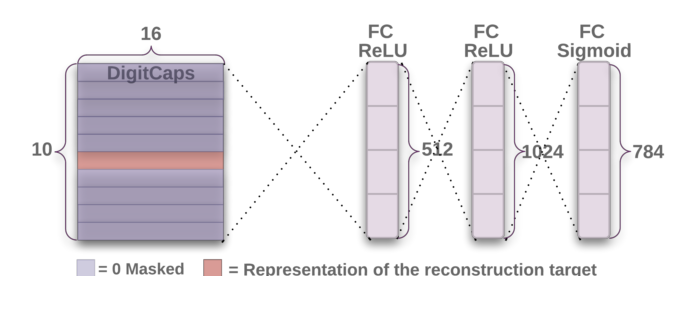


Figure Decoder structure

* Decoder is used as a regularizer.
* Input: from correct DigitCap
* Output: 28x28 image. It reconstructs the digit.
* #Params: 161\*512+513\*1024+1025\*784=1411344
* Loss: Euclidean distance between original digits and reconstructed digits.

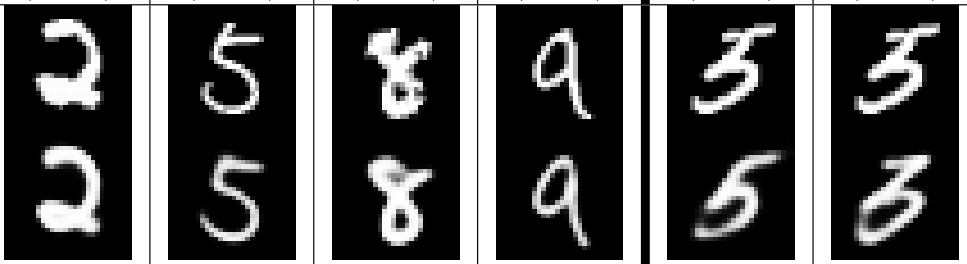


Figure Top row: original digits. Bottom row: reconstructed digits.