Introduction to Capsule Network

Qifan Yang

Minerva Schools at KGI

Convolutional Neural Network and how to build it

We have learned the basic knowledge of neural network. To get us started, here is the mathematical representation of the formula for a layer to pass the information to the next layer:

$$a_j = \sum_i w_{ij} x_i + b_j, \qquad h_j = f(a_j)$$

The neuron a_j sums up the multiplication of the activation x_i and the weight w_{ij} and adds an addition bias b_j. An activation function f is used to 'squish' the sum down to a certain range, usually between 0 and 1. Common activation functions include Sigmoid, ReLU, tanh and softmax.

Rewrite the formula in a more concrete way, we get

$$a^{(p)} = f\left(W a^{(p-1)} + b\right).$$

Different types of layer could be constructed in neural network: In CNN, convolutional layers are applied to summarize features from a picture. As introduced in class, different convolution matrices could introduce different filter effects on images and find out patterns within edges, color or gradient.

```
#------Gaussian Blur Matrix------#
mat = np.zeros((matrixsize, matrixsize))
mid = matrixsize // 2

sigma = float(radius)
for i in range(matrixsize):
    for j in range(matrixsize):
        mat[i][j] = np.exp(-((i - mid)**2 + (j - mid)**2) /
                           (2 * sigma**2)) / (2 * np.pi * sigma**2)
kernel = mat

#-------Matrix Convolution-------#
ans = []
for i in range(3):
    chan = part[:, :, i]
    ans.append(convolve2d(chan, kernel, fillvalue = 255)
               [mid:-mid, mid:-mid])
```

Fig.1. Example of applying Gaussian Blur using 2D Gaussian matrix. (Source: LBA)

Another type of layers used in CNN is pooling layer, the most popular of which being max pooling. It partitions the input image into rectangles and gathers all the maximum values

in each rectangle. Two paramaters, pool size and stride, corresponds to the size of rectangle and the distance between each rectangle.

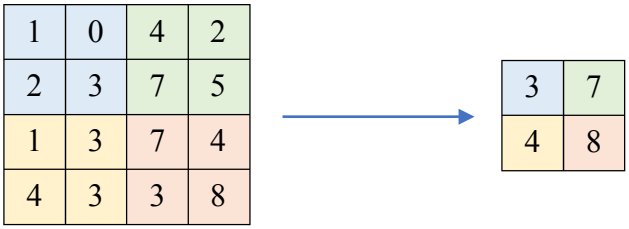


Fig.2. Example of max pooling with pool size of 2x2 and stride of 2.

The model VGG19 used 16 convolution layers and 5 max pooling layers to build up a model, and classifies object into 1000 object categories with 77.8% accuracy. (Simonyan & Zisserman, 2015). Amazing, right?

Pitfall of CNN

In Convolutional network, deeper layers detect simple features; higher layers could combine those simple features together; The final dense layers combine the features even higher and produces the classification result.

However, neither the weighted sum combination nor the pooling layers could clearly express the spacial relationship between simple featuer explicitly. I drew a hilarious example to express this deficiency:

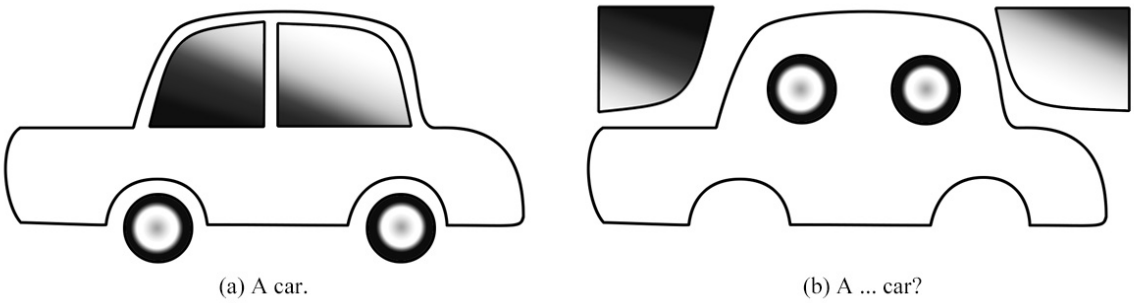


Fig.3. Deficiency of CNN within spacial hierarchies.

In Fig.3, the CNN would recognize separate features such as windows with gradient colors or a tire, but would find it hard to represent the relative direction of these objects, and the correct spacial hierarchy between these objects within a trained model.

The function, max pooling, extracts higher level feature by pulling out the maximum weight in an area. Yet this causes the neural network to ignore important features such as the relative position of objects. As Hinton G. mentioned in his reddit AMA thread in 2014,

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

Capsule network

Hinton holds a different idea for image recognition: the brain deconstructs hierarchical representation of the world from visual input and match them with pattern. (Hinton. et al., 2012) The model doesn't depend directly on the angles between features, since projection of 3D objects in 2D space are likely to have different patterns compared to the original data.

In 3D graphics, relationships between 3D objects can be represented by a so-called pose, which is in essence translation plus rotation. In order to apply the concept in neural network, the activation of each neuron need to be a vector u_i, rather than a scalar x_i. The word 'capsule' represents a group of neurons using activity vectors, and replaces neurons as the new unit in Capsule Network.

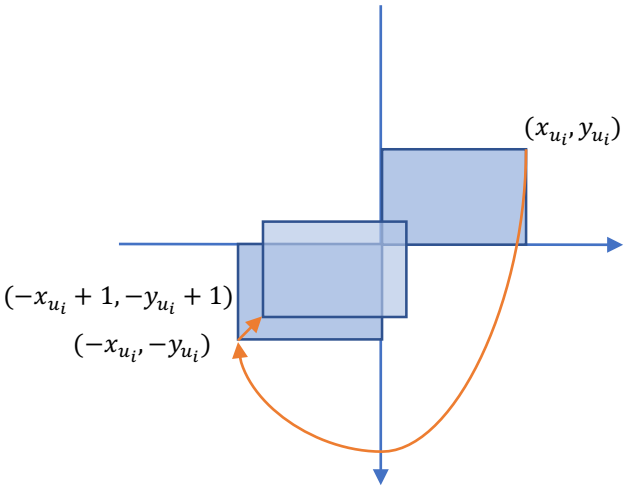
How do we compute the input and output of a capsule? Or, given a layer of capsules (u_i being one of the capsules), how do we compute the capsule v_j in the next layer?

(Note for the following steps: Even with similar symbols and functionalities, the structure of Capsule Network is **different** from that of the traditional convolutional neural network. It would be best to start from zero and compare the two structures after going through the process.)

Matrix multiplication and Affline transform

$$\hat{u}_{j|i} = W_{ij}u_i$$

The W matrix captures the core idea of hierarchical relation - translation plus rotation. Suppose that u_i is a two-dimensional vector (x_{u_i}, y_{u_i}), the affline transformation reflecing the vector about origin and moves it by $(1, 1)$ could be calculated using the following matrix multiplication:



$$\begin{bmatrix} x'_{u_i} \\ y'_{u_i} \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{u_i} \\ y_{u_i} \\ 1 \end{bmatrix}$$

Fig.4. Example of Affline Transformation

Both the vector and matrix are expanded by 1 in all dimensions, the row of which are fixed to 1 and $[0, 0 \ldots 1]$. The top left part of the matrix is the transition matrix for linear transformation, and the rightest column without the value at the bottom left corner represents the vector addef in the following translation step. Affline Transformation completes both job

neatly and broadens the functionality of linear transformation simply by increasing both dimensions of the matrix by 1.

Scalar weighting and Dynamic Routing

$$s_j = \sum_i c_{ij} \hat{u}_{j|i}$$

In the author's words: "The c_{ij} are coupling coefficients that are determined by the iterative dynamic routing process."

Again, what is c_{ij}? It is a non-negative scalar, and the sum of all c_{ij} for each lower capsule i equals to 1. Or, C is a 2-dimensional matrix, and its dimension are aligned to number of capsules in each layers. Unlike the updating process of W (which is back propagation after each prediction), C is calculated in real time within each capsule. Here is the procedure stated in the original article:

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}$, 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: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `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 $\mathbf{v}_j$

Fig.5. Routing Algorithm. (Source: Hinton et al., 2017)

Here are some of my ideas to help understand the algorithm:

(1) Dynamic routing wraps all the calculation steps for each neuron up, and iterates several times to get a decent C matrix. The squash function will be mentioned later.

(2) A vivid explaination of C matrix would be the following question: 'which capsule would prefer $\hat{u}_{j|i}$'s information regarding the information from all $\hat{u}_{j|i}$s?'.

In the picture below, the value of c_{11} is reduced as $\hat{u}_{1|1}$ does not match the direction of v_1; Meanwhile, the value of c_{12} is boosted as it does match the direction of v_2. The weight adjustment of c_{ij} in a single iteration would 'tell' $\hat{u}_{j|i}$ to what extent $\hat{u}_{2|1}$ should send its value to capsule v_j in the next layer.

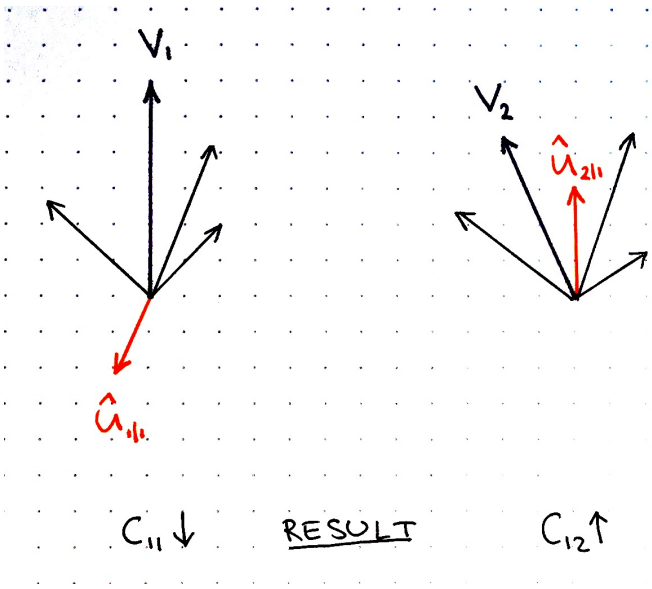


Fig.6. Weight adjustment of c_{11} and c_{12} according to the higher level vectors v_1 and v_2. (Source: Pechyonkin, medium)

(3) A **not-so-appropriate** analogy for b and c: Think of b as the parameter of Dirichlet distribution, and c as one single sample generated from the Dirichlet Distribution. This is not true since the value of c is fixed given b. Yet here is why the analogy holds:

- b has a prior of zeros;
- b changes according to the 'likelihood' of $\hat{u}_{j|i}v_j$, and sums up its value.
- Similar to the samples of the Dirichlet Distribution, the sample c_{ij} from softmax also sums up to 1. (softmax: $c_{ij} = \frac{e^{b_{ij}}}{\sum_k e^{b_{ik}}}$)

(4) A new v_j is generated during each iteration. Then, a new C matrix is generated in the next iteration, acting as the parameter for a new v_j value.

(5) The number of iterations r need to be controlled carefully, or the algorithm would be subject to overfitting[1].

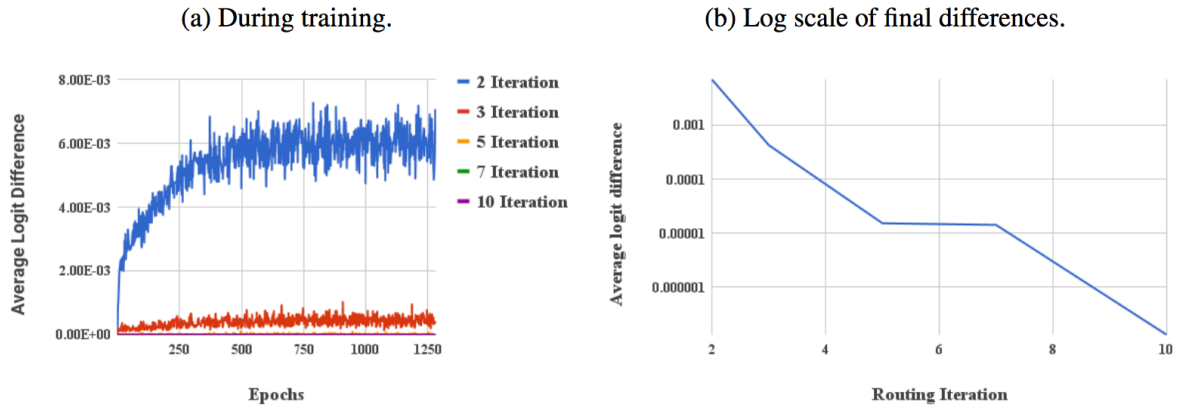


Fig.7. Difference between each routing iterations. (Source: Hinton et al., 2017)

As shown in Fig.7, after 500 epochs of training on MNIST the average change is stabilized, and the difference decreases linearly in log scales as the number of iteration grows. Adding more routing iterations would lead to greater network capacity as the trained samples being enhanced, but also tend to overfit the training dataset. An iteration number of 3 is suggested in in the paper after multiple trials.

To summarize Dynamic Routing in one sentence, after we calculate the vector $\hat{u}_{j|i}$ for each u_i, we obtain a relatively acceptable c_{ij} value through v_j, calculate v_j based on c_{ij}, and iterate the process to enhance learning.

Squashing, the activation function for vector

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

[1] #overfitting, as described here.

Squashing function is the activation function for vector. To break the formula down, $\frac{s_j}{\|s_j\|}$ represents the direction of v_j as a unit vector; $\frac{\|s_j\|^2}{1+\|s_j\|^2}$ represents the length of v_j as a scalar. Together the formula would ensure:

(1) v_j is a vector with length smaller than 1;

(2) v_j keeps the direction information from s_j.

CapsNet Architecture for MNIST

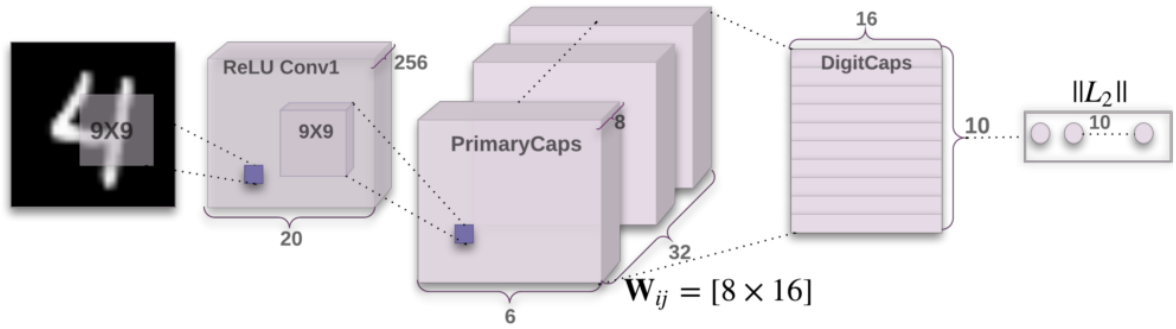


Fig.8. CapsNet Architecture (Source: Hinton et al., 2017)

Layer Name	Layer Type	Capsule Dimension	Layer Shape	Description
Input	FC[2]	/	(28, 28, 1)	Raw images in grey scale
Conv1(ReLU)	Convolution	/	(20, 20, 256)	Stride = 1, Kernel = (9, 9)
PrimaryCap	Capsule	8	(6, 6, 32)	Stride = 2, Kernel = (9, 9)
DigitCaps	Capsule	16	(10)	Shape of W_{ij} = (8, 16)
Length	/	/	(10)	Discard everything but length

Table.1. CapsNet Architecture

[2] FC layer stands for 'fully connected' layer.

(1) As the padding is set to 'valid', the width and height of layer shrinks as we apply the convolution kernel in ReLUConv1 and PrimaryCap.

(2) No routing happens between ReLUConv1 and PrimaryCap: the third dimension shrinks to one eighth of its original size to offer the capsule extra dimensions. The shrinking process helps with the transform to Capsule Network.

Extra Decoder Design After Digitcaps

Layer Name	Activation Function	Capsule Dimension	Layer Shape	Description
DigitCaps	/	16	(10)	
FC1	ReLU	/	(512)	Splits capsule dimensions
FC2	ReLU	/	(1024)	
FC3	Sigmoid	/	(784)|(28, 28)	Reconstruction output layer

Table.2. Decoder Architecture

The decoder is used to 'encourage the digit capsules to encode the instantiation parameters of the input digit' (Hinton et al., 2017). The decoder aims to minimize the sum of squared differences (In my opinion it is just MSE) between decoder output and the given pixels. Fig.10. presents samples of real digit images and generated digit images.

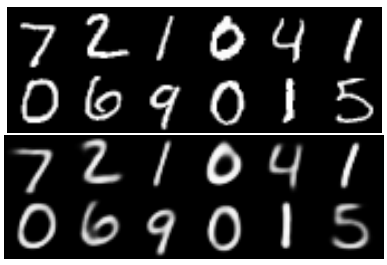


Fig.9. Top: Real digit images; Bottom: Generated digit images.

Summarization

A traditional convolutional model with a similar number of parameters achieved 99.22% accuracy, while an under-trained CapsNet achieved 99.23% accuracy on the expanded MNIST test set. While there seems to be no improvement, the CapsNet achieved 79% accuracy on the affnist test set compared to 66% accuracy by traditional convolutional model. (Hinton et al., 2017) As the CapsNet contains intrinsic affine transformation system, it matches the generation process of affNIST and achieves better accuracy[3]. This is not considered as overfitting, but an approach to close the distance with real world application in image recognition.

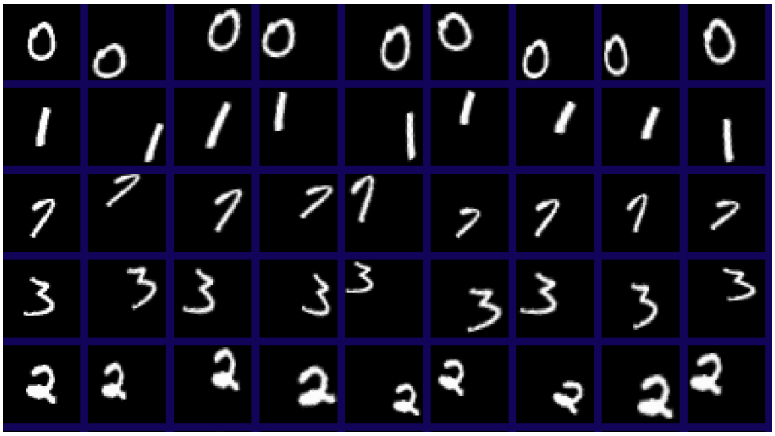


Fig.10. Samples of affNIST data set. affNIST takes images from MNIST and applying various reasonable affine transformations to them. The left column shows the original MNIST digit (centered in a 40x40 image), and the other columns show transformed versions. (Source: affNIST)

Though Hinton raised the concept of Capsule Network years ago (Hinton et al. 2011), the Dynamic Routing methodology manage to transform lowers patterns into higher patterns

[3] #modelmetrics: Precision and Recall are not the best metrics here, since we focus on the general capability of recognition rather than 'How many selected 0s are 0s' for precision or 'How many 0s are selected among all 0s in data' for recall. Accuracy calculates the ratio of true positives and true negatives within all datas, and is considered as the best metrics among the three for MNIST and affNIST.

between two capsule layers. The code attempting to train an Capsule network using aforementioned architecture is attached in the Appendix.

Reference

Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in Neural Information Processing Systems* (pp. 3857-3867).

Pechyonkin M. (2017). Understanding Hinton's Capsule Networks. Medium. Retrieved from https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

Hinton G., Krizhevsky A., Jaitly N., Tieleman T. & Tang Y. (2012). Does the Brain do Inverse Graphics? Department of Computer, Science University of Toronto. Retrieved from http://helper.ipam.ucla.edu/publications/gss2012/gss2012_10754.pdf

Guo, X., (2013). CapsNet-Keras. GitHub repository, Retrieved from https://github.com/XifengGuo/CapsNet-Keras

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Hinton, G. E., Krizhevsky, A., & Wang, S. D. (2011, June). Transforming auto-encoders. In *International Conference on Artificial Neural Networks* (pp. 44-51). Springer, Berlin, Heidelberg.

affNIST dataset. (2013). Toronto University. Retrieved from http://www.cs.toronto.edu/~tijmen/affNIST/

Hinton, G. (2014). AMA Geoffrey Hinton. Reddit. Retrieved from https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/clyj4jv/